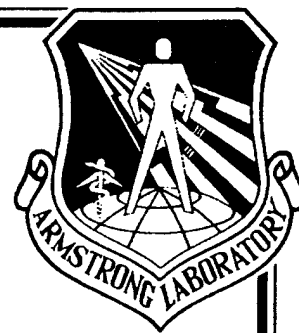


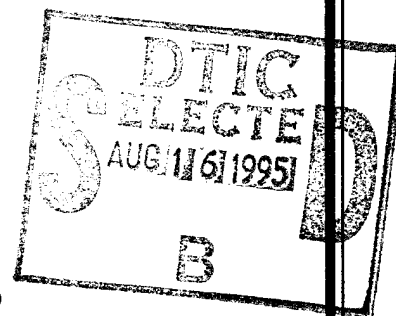
2

AL/HR-TP-1995-0012



**ALTERNATIVE ANALYSIS FOR  
COMPUTATIONAL HOLON ARCHITECTURES**

**Bernard P. Zeigler  
Sankait Vahie  
Doo-Hwan Kim**



**Artificial Intelligence and Simulation Group  
Department of Electrical Computer Engineering  
College of Engineering  
The University of Arizona  
Tucson AZ 85721**

**HUMAN RESOURCES DIRECTORATE  
LOGISTICS RESEARCH DIVISION  
2698 G Street  
Wright-Patterson Air Force Base, Ohio 45433-7604**

**June 1995**

**Final Technical Paper for Period February 1993 to February 1994**

**Approved for public release; distribution is unlimited**

**AIR FORCE MATERIEL COMMAND  
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-7022**

19950814 100

DTIC QUALITY INSPECTED 1

ARMSTRONG  
LABORATORY

365


## NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this paper and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

  
MICHAEL J. YOUNG  
Contract Monitor

  
BERTRAM W. CREAM, Chief  
Logistics Research Division

Approval For	
USE	<input checked="" type="checkbox"/>
USE	<input checked="" type="checkbox"/>
(unclassified)	<input checked="" type="checkbox"/>
Justification	
By	
Distribution	
Availability Codes	
Availability	
Spec	Spec
A-1	

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE June 1995	3. REPORT TYPE AND DATES COVERED Final - February 1993 to February 1994		
4. TITLE AND SUBTITLE Alternative Analysis for Computational Holon Architectures		5. FUNDING NUMBERS  C - F33615-91-D-0009 PE - 62205F PR - 1710 TA - 00 WU - 60		
6. AUTHOR(S) Bernard P. Zeigler Sankait Vahie Doo-Hwan Kim		8. PERFORMING ORGANIZATION REPORT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)  Artificial Intelligence and Simulation Group Department of Electrical Computer Engineering College of Engineering, University of Arizona Tucson AZ 85721		10. SPONSORING / MONITORING AGENCY REPORT NUMBER  AL/HR-TP-1995-0012		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)  Armstrong Laboratory Human Resources Directorate Logistics Research Division 2698 G Street Wright-Patterson AFB, OH 45433-7604		11. SUPPLEMENTARY NOTES  Armstrong Laboratory Technical Monitor: Michael J. Young, AL/HRGA, DSN 785-8229		
12a. DISTRIBUTION / AVAILABILITY STATEMENT  Approved for public release; distribution is unlimited		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words)  This effort investigated programming languages, computer processors, and simulation techniques capable of supporting the development of an advanced framework for human performance process model research. The framework, the holon cognitive architecture, conceptualizes the mind as consisting of a set of interactive agents. These agents are abstract models of neuronal activity occurring in the brain. This document contains an analysis of the computation requirements needed to simulate human performance process models based upon the holon cognitive architecture and suggest a growth path based upon a heterogeneous computing environment.				
14. SUBJECT TERMS  parallel processing distributing computing human performance modeling object-oriented simulation		15. NUMBER OF PAGES 114		16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT  Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE  Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT  Unclassified	20. LIMITATION OF ABSTRACT  SAR	

## PREFACE

This effort investigated programming languages, computer processors, and simulation techniques capable of supporting the development of an advanced framework for human performance process model research. The framework, the holon cognitive architecture, conceptualizes the mind as consisting of a set of interactive agents. These agents are abstract models of neuronal activity occurring in the brain. This document contains an analysis of the computation requirements needed to simulate human performance process models based upon the holon cognitive architecture and suggest a growth path based upon a heterogeneous computing environment.

The reported work was conducted between February 1993 and February 1994 supplements and extends the results of an earlier delivery order (Delivery Order No. 1: Computational Cognitive Models) administered under the same USAF contract (No. F33615-91-D-0009) and conducted a year earlier. The latter effort formulated an HPP model based on theoretical and empirical findings in psychology and cognitive science that could be used to support operability investigations of new systems.

The authors wish to thank the contract monitor, Mr. Michael Young of Armstrong Laboratory's Human Resources Directorate, for his constructive criticism and enthusiastic support of the work.

# Contents

<u>CHAPTER</u>	<u>PAGE</u>
List of Figures	5
List of Tables	7
Summary	8
<b>1 Introduction</b>	<b>10</b>
1.1 Limitations of Human Performance Modeling for Operability Analysis . . .	10
1.2 Human Performance Process Models . . . . .	12
1.3 Other Human Performance Process Related Models . . . . .	12
1.3.1 Newell's Soar . . . . .	13
1.3.2 Anderson's Adaptive Control of Thought (ACT-Star or ACT*) . . .	13
1.4 Limitations of Existing Human Performance Process Models . . . . .	14
1.4.1 Behavioral Limitations . . . . .	14
1.4.2 Competence Limitations . . . . .	14
1.5 The Holon Concept . . . . .	16
1.6 Holon Architectures for Human Performance Process Models . . . . .	17
1.7 Conclusion . . . . .	18
<b>2 Metaphors for Human Performance Process Model Development</b>	<b>19</b>
<b>3 Human Performance Process Functionality Requirements</b>	<b>23</b>

## Contents (continued)

<u>CHAPTER</u>	<u>PAGE</u>
<b>4 General Requirements for Languages/Environments</b>	<b>28</b>
4.1 Requirements for Language/Environment Support of Human Performance Process Modeling . . . . .	29
4.2 Candidate Languages/Environments for Human Performance Process Model Development . . . . .	30
<b>5 Evaluation of Languages/Environments</b>	<b>32</b>
<b>6 Review of Parallel Processing Concepts</b>	<b>35</b>
6.1 Dimensions of Architecture . . . . .	35
6.2 Dimensions of Problem Decomposition . . . . .	36
6.3 Examples of Problem/Architecture Pairings . . . . .	37
6.3.1 Data Parallel Model . . . . .	37
6.3.2 Artificial Neural Nets . . . . .	38
6.4 Heterogeneity of Processing in Performance Process Models . . . . .	38
<b>7 Hardware Architecture Requirements</b>	<b>43</b>
<b>8 Hardware Architectures</b>	<b>45</b>
8.1 Single Workstations . . . . .	45
8.2 Networks of Workstations . . . . .	45
8.3 High Performance Supercomputers . . . . .	47
8.4 Heterogeneous Computing Environments . . . . .	48
<b>9 Evaluation of Platforms</b>	<b>51</b>
9.1 Scalability Performance Metrics . . . . .	51
9.2 High Performance System Metrics . . . . .	52
9.3 Workstation Price & Performance Metrics . . . . .	53

## Contents (continued)

<u>CHAPTER</u>	<u>PAGE</u>
9.4 Price/Performance Baselines for Computer Hardware . . . . .	54
9.5 Entry-level System Metrics . . . . .	55
9.6 Recommendations . . . . .	56
<b>10 Hierarchical Evolution of Human Performance Process Models &amp; Platforms: Successive Approximation</b>	<b>59</b>
<b>11 Follow-on Investigation</b>	<b>67</b>
<b>Appendix A. Human Performance Process Metaphors</b>	<b>69</b>
<b>Appendix B. Synopses of Selected Languages/Environments</b>	<b>75</b>
<b>Appendix C. Neural Networks: An Overview</b>	<b>78</b>
<b>Appendix D. Simulation Concepts</b>	<b>81</b>
D.1 Real-Time Simulation . . . . .	81
D.2 Approaches to Distributed Simulation . . . . .	82
D.2.1 Conservative Approach . . . . .	82
D.2.2 Optimistic Approach . . . . .	83
D.3 Discrete Event System Specification . . . . .	85
D.3.1 Hierarchical, Modular Structure . . . . .	85
D.3.2 The Basic Model . . . . .	85
D.3.3 The Coupled Model . . . . .	86
D.3.4 The System Entity Structure . . . . .	87
D.3.5 The Abstract Simulator . . . . .	87
<b>Appendix E. Examples of Human Performance Process Hierarchical Decomposition</b>	<b>92</b>
<b>Appendix F. Scalable Coherent Interfaces</b>	<b>96</b>

## Contents (continued)

<u>CHAPTER</u>	<u>PAGE</u>
Appendix G. Synopses of Selected High Performance Parallel Machines	98
Appendix H. Glossary of Acronyms	102
References	105

## List of Figures

<u>FIGURE</u>	<u>PAGE</u>
1.1 A Holarchy . . . . .	17
2.1 Possible Paths for Human Performance Process Model Creation . . . . .	21
6.1 Numerical Aerodynamics Simulation Results for Embarassingly Parallel Benchmarks . . . . .	40
6.2 CM2: Numerical Aerodynamics Simulation Benchmark Results . . . . .	41
6.3 Human Performance Process and Architectures . . . . .	42
8.1 Heterogeneous Computing Environment . . . . .	50
9.1 High Performance Systems Metrics . . . . .	52
9.2 Workstation Price & Performance Metrics . . . . .	53
9.3 Price/Performance Baselines for Computer Hardware . . . . .	54
9.4 Entry-Level Systems Metrics . . . . .	55
9.5 Scalability Performance Metrics of Data Parallel Platforms . . . . .	57
9.6 Scalability Performance Metrics of Multiple Instruction Multiple Data (MIMD) Platforms . . . . .	58
10.1 Human Performance Process Model Space . . . . .	60
10.2 "Continuous" Mapping: Models to Architectures. . . . .	61
10.3 Human Performance Processing Model Prototype . . . . .	62
10.4 Operations on the System Entity Structure . . . . .	63
10.5 Evolutionary Network of Computing Nodes . . . . .	64
10.6 Hierarchical Evolution I . . . . .	65

## List of Figures (continued)

<u>FIGURE</u>	<u>PAGE</u>
10.7 Hierarchical Evolution II . . . . .	66
C.1 A Back Propagation Neural Network . . . . .	80
D.1 Requirements for Real-Time Processing . . . . .	82
D.2 Class Hierarchy of Discrete Event System Specification in Scheme . . . . .	86
D.3 Discrete Event System Specification: Concept of Modularity . . . . .	88
D.4 Discrete Event System Specification Hierarchical Abstract Simulator Structure	88
D.5 Algorithm of Root-Coordinator . . . . .	89
D.6 Algorithm of Coordinator . . . . .	90
D.7 Algorithm of Simulator . . . . .	91
E.1 Distributed Cognitive Simulation Architecture (Root Node) . . . . .	92
E.2 Cognitive Subsystem Decomposition . . . . .	93
E.3 Cognitive Subsystem 3 Decomposition . . . . .	93
E.4 Sensory Subsystem Decomposition . . . . .	94
E.5 Sensory Subsystem 3 Decomposition . . . . .	94
E.6 Motor Subsystem Decomposition . . . . .	95
E.7 Motor Subsystem 3 Decomposition . . . . .	95
F.1 Evaluation of Network Interconnection Interfaces . . . . .	97

## List of Tables

<u>TABLE</u>	<u>PAGE</u>
2.1 Contributions of Metaphors to HPP Modeling & Barriers to Application . . .	22
5.1 Evaluation of AI Languages/Environments with respect to HPP Functionality	33
5.2 Evaluation of Simulation Environments with respect to Model Development	34

## Summary

**Chapter 1. Introduction:** The background and objectives of the project is presented to set the context for further discussion.

**Chapter 2. Metaphors for Human Performance Process Model Development:** An approach adopted to develop aspects of functionality for Human Performance Process (HPP) models, such aspects being intended to serve as a basis for specifying the requirements of software and hardware support systems is presented in this chapter. Metaphors are classes of models drawn from a variety of literature that reveal information about what structures and behaviors might be included in an HPP model. Detailed analysis of each metaphor is provided in an appendix. The chapter concludes with a brief discussion of possible heuristic use of the metaphors for HPP model development.

**Chapter 3. Human Performance Process Functionality Requirements:** This chapter presents the results of compounding the aspects generated from the metaphors of the previous chapter. To act as requirements, such aspects are grouped under a small number of descriptors (e.g., intelligence/competence, performance attributes, built-in primitives, etc.) at three levels (neural, symbolic, and functional).

**Chapter 4. General Requirements for Languages/Environments:** This chapter enumerates requirements that languages/environments should meet to support HPP modeling and simulation and that they are of a more general nature than the functionality requirements developed in the previous chapter.

**Chapter 5. Evaluation of Languages/Environments:** Representative languages and environments are evaluated for their potential to support HPP model development using the criteria developed in two previous chapters. Synopses of these systems are provided in an appendix. Results show that systems typically support only a limited subset of the identified requirements.

**Chapter 6. Review of Parallel Processing Concepts:** For reference in subsequent chapters, the terminology of parallel processing and principles for relating problems and architectures are reviewed.

**Chapter 7. Hardware Architecture Requirements:** Although the original focus of the task was to characterize the functional capabilities of high performance parallel computers, it was soon realized that supporting a mix of computing styles matching

individual component requirements of HPP models provided a means to escape the restraints of a single style platform. Also, analysis revealed that it would be critical to be able to smoothly evolve HPP modeling support from a single workstation to a higher performance platform and that real-time operation would be necessary for man-in-the-loop simulation. As a consequence, this chapter develops hardware requirements based on heterogeneity, evolvability, and real-time simulation potential.

**Chapter 8. Hardware Architectures:** In view of the requirements in the previous chapter, four levels of computing environments are considered: single workstations, homogeneous networks of workstations, high performance machines, and heterogeneous computing networks. Software support for networks is also discussed. This chapter briefly reviews the salient architectural features of each of these categories. Synopses of representative platforms are presented in an appendix.

**Chapter 9. Evaluation of Platforms:** To evaluate representative platforms performance benchmark data is extracted from current literature and is presented in graphs and in tabular form. The objective is to portray clearly the realizable scalability characteristics and performance and cost ratios of representative systems. The entry level costs are also presented to assess capital outlay. Of the alternatives, a network of workstations has the best evolvability characteristics and surprisingly, a performance/cost ratio rivalling the best of the supercomputers. However, whether real-time and heterogeneity requirements of HPP simulation could be met by a homogeneous network is unclear. This uncertainty lead to the consideration of an approach to HPP model and platform co-evolution that could support such requirements, presented in chapter 10.

**Chapter 10. Hierarchical Evolution of Human Performance Process Models and Platforms: Successive Approximation:** An approach to model development based on hierarchical decomposition that supports co-evolution of model development and platform expansion is recommended. A demonstration shows how this approach may offer an effective and flexible approach to meeting heterogeneity requirements, especially coupled with the demands for real-time model operation imposed by man-in-the-loop simulations. It is concluded that networks of workstations are able to support such hierarchical expandability.

**Chapter 11. Follow-on Investigation:** This chapter discusses the next research steps that would be required to investigate alternative designs for supporting hierarchical HPP model evolution and establishes their feasibility.

# Chapter 1

## Introduction

The goal of United States Air Force (USAF) operability analysis is to ensure that human performance requirements are adequately incorporated into the early stages of system design. Alternative system designs differ as to the amount of automation-aiding technology, the crew-size requirements, or the skill-level requirements. A typical study begins with a "concept of operations" (CONOPS) which defines the method by which the system will be employed, the crew levels required, and the procedures the operators will follow. Representative test scenarios, consistent with CONOPS directives, are developed to include both normal and emergency operation. Testbeds under development are intended to enable operational personnel to "fly the system" while it is being designed, and thus provide early feedback to the designers. Human performance models integrated into this process serve as team members in a multicrew environment. Such models interact with the humans-in-the-loop through voice generation and recognition systems, as well as the system interface. In this capacity, human performance models replace expensive human study participants and reduce experimental variability by providing consistent behavioral replication across simulation trials.

### 1.1 Limitations of Human Performance Modeling for Operability Analysis

Ideally, existing models of human operators of complex dynamic systems could be applied to operability analysis. However, human performance modeling (HPM) has been the subject of intensive studies [Elkind-89, Baron-90], which document the problems that prevent the reuse of some two hundred models developed in last decade. As summarized by Young [Young-93], these problems are as follow.

- *Lack of coverage in modeling objectives and consequently of essential processes.* Traditionally, HPMs were designed for situations in which an operator is engaged in a single task controlling a machine. However, in today's high performance environments, the system operators are engaged in complex tasks requiring multiple inputs and outputs.

They must entertain multiple goals and are subject to frequent priority-demanding interruptions. Moreover, with the increased complexity of new systems, differences in individual skills and styles will play an increasingly important role in man/system performance.

Existing models do not provide a complete description of human perceptual and cognitive performance. There are significant gaps in the availability of models for several stages of information processing. Most models include perceptual or psychomotor components of behavior but do not account for cognitive behavior. Finally, individual differences have largely been ignored in favor of normative or average indices.

- *Multiplicity of alternative formulations for processes that are covered.* Models built by different investigators differ in strategies for processing information (e.g., employing an algorithmic or heuristic search) and on the specifics of the inferencing techniques, which specify the algorithm or type of heuristic search. Knowledge representation schemes typically differ on the format, content, and organization of representations (e.g., rules, frames, semantic nets). These model differences are snapshots of the theoretical differences being debated by scientists. They reflect fundamental differences in beliefs concerning the way information is represented and employed in cognition.
- *Incompatibility of micromodel interfaces and resolution levels.* Lack of agreement between inputs and outputs of individual micromodels preclude their interconnection into larger units. Linkages between models representing different stages within a modality (e.g., vision) are weak, and linkages between modalities (e.g., vision and cognition) are essentially nonexistent. Differences in input and output for a stage may reflect differences in whether a micromodel expects to receive (or to produce) numerical data (e.g., summation of a spreading activation vector) or symbolic messages (e.g., a description of a class and its potential interactions).
- *Lack of validation and independent evaluation.* The more complex the model, the more difficult and costly it is to validate. The lack of validated models is the main factor preventing a wider use of HPMs in system design. Most complex HPMs have not been widely used or subjected to independent evaluation.
- *Absence of documentation of underlying assumptions and theoretical bases.* As models become more complex, the potential for misuse and misunderstanding of underlying assumptions and theoretical bases grows. Documentation describing the embedded data and software requirements necessary for model usage is generally absent.

The limitations of contemporary HPM models, both individually and collectively, motivate the development and validation of more capable models that can be archived in a reusable model base and "compiled" as needed to support diverse operability study requirements. Support of such long term model development requires appropriate methodologies, software environments, and hardware platforms. The task of the rest of the report is to discuss possibilities and recommend some of the possibilities for further consideration.

## 1.2 Human Performance Process Models

Human Performance Process (HPP) models attempt to emulate human behavior through the simulation of specific human information processing attributes and processes. The goal is to permit psychological principles and data to influence the design of new human-operated performance-critical systems. Two possible applications of HPP models in their current state are as follows:

- *System Modeling Testbeds.* These testbeds support operability analysis, and human-in-the-loop simulations and evaluate the design options in multi-crew environments.
- *Cognitive Science Research.* This research tests psychological theories and compares alternative models with respect to performance impact.

An example of an HPP model was constructed by BBN Systems & Technology. It consisted of five subcomponents [Young-92].

- *Visual.* This component contains two types of visual processing.
  - *Active gaze:* Focused and directed movement to a specific target.
  - *Monitoring:* An unguided scanning process.
- *Auditory.* This component incorporates model communication protocols employed by human operators. Modeling parameters include: communication bandwidths, and short-term and long-term memory limits of human processing capabilities.
- *Cognitive.* The cognitive component depicts cognitive activity via procedural representation of IF...THEN statements and an inference engine. In other words, the cognitive subcomponent is modeled as an expert system model [Hayes-Roth-83].
- *Psychomotor.* This component uses Fitts' formulation that relates movement time, distance, and accuracy to provide a probability of error.
- *Updatable World Representation.* Each subcomponent model contains an individually defined, updatable world representation that is composed of rules for decision making, an awareness of external events (as seen through the operator's perceptual processes), and a declarative description of the world.

The expert's knowledge is represented as a symbol structure with rules for manipulating knowledge (heuristics). The control strategy, in part, is based on the psychological model of human memory.

## 1.3 Other Human Performance Process Related Models

While a number of models have been developed to model specific cognitive processes, two models have gained widespread recognition in their efforts to develop a unified theory of

human cognitive processing: Anderson's Adaptive Control of Thought (ACT) and Newell's Soar [VanLehn-91]. For the purpose of better understanding the current state of HPP modeling these models, their orientations, and their underlying structures [Hudlicka-92] are discussed briefly below.

### 1.3.1 Newell's Soar

*Soar* [Newell-90] is a first attempt at modeling the learning process in humans based on Newell's "unified theory" of cognition. Implemented as an architecture for cognitive processing, *Soar*'s knowledge is supplied as *problem spaces* representing domain-specific knowledge and heuristics. *Soar* attempts to implement Newell's **Problem Space Hypothesis** in which the problem space, consisting of states, is mapped onto other states by applying operators until the desired goal state is reached (much like Newell's GPS: General Problem Solver).

Using a process called *chunking*, a sequence of operations (production rules) previously applied with favorable results, are grouped together to produce a new operator. This grouping represents an effort to approximate learning through experience.

In the *Soar* architecture, all knowledge is represented in the form of rules, while memory is incorporated into the rules used by the system (Long-Term memory) and working memory (Short-Term memory). Thus, *Soar* is unable to differentiate between various kinds of knowledge and memory (procedural and declarative knowledge, semantic and episodic memory, etc.).

In terms of human performance, *Soar* focuses on a unified theory to explain cognitive learning. The level of resolution at which *Soar* functions enables it to explain behavior that may be categorized as either cognitive (a few seconds) or rational (a few minutes to an hour).

### 1.3.2 Anderson's Adaptive Control of Thought (ACT-Star or ACT\*)

ACT\*, a cognitive model developed by Anderson to simulate high-level memory-related processing [Anderson-90], is an attempt to identify the underlying architecture that forms the basis for high-level cognitive processing. It differs from *Soar* because it defines, and distinguishes between, both procedural and declarative memory.

In ACT\*, procedural and declarative knowledge are represented as production rules and semantic nets, respectively. The process of transforming declarative ("factual") information into procedural knowledge is modeled in depth. In addition to differentiating between procedural and declarative knowledge, ACT\* distinguishes between long and short-term (or "active") memory.

ACT\* long-term memory is represented in the form of object-attribute-value tuples. Declarative knowledge contains multiple data-type definitions: images, sequences, and so forth. Processing is accomplished by *spreading activation* in the semantic net by applying the

procedural knowledge, whereby nodes and relations in the semantic net can be added (or deleted).

ACT\* takes *Soar's* process of chunking one step further. Chunks are composed of two to five elements of declarative memory, classified into three categories (prepositions, spatial links, and temporal links) for use at different times in the processing of information.

Goal-directed behavior is modeled to represent controlled (conscious) processing of behavior, as opposed to data-directed (learned/automated) behavior. Equipped with sophisticated conflict resolution strategies, ACT\* is capable of modeling high-level cognitive processing (especially memory retrieval and skill acquisition) with significant accuracy. However, the domain and scope of the modeled behavior is limited.

## 1.4 Limitations of Existing Human Performance Process Models

Several facets of HPP models need continued research and development [Young-92, Young-93]. These facets can be divided into *Behavioral* and *Competence* issues. Behavioral issues related to HPP models arise from their inability to predict human performance. Competence limitations arise from the lack of sophistication in depicting the information processing models in contemporary HPP models.

### 1.4.1 Behavioral Limitations

Behavioral issues that need to be addressed by HPP models include the following.

- Attention limitations due to high tasks demands.
- Skill level differences.
- Concurrent task performance.
- Context and task prioritization.

### 1.4.2 Competence Limitations

Three dimensions in which sophistication is necessary have been identified.

- *Top-Down and Bottom-Up Data Processing(DP)*. In bottom-up DP, information is transformed from small perceptual chunks (data driven processing) into a more abstract perceptual information. An example of bottom-up processing is *pattern recognition*, where each atomic pixel is related to its neighbors and a pattern is perceived. Top-down DP uses a situational context and general world knowledge to drive the processing and interpretation of information (conceptually driven). An example is

the *planning* of a high-level task, where the task is broken down into sub-task and each sub-task is further sub-divided.

Both top-down and bottom-up processing capabilities need to be incorporated into human performance simulators. Top-down DP is used to guide bottom-up DP in the recognition of stimuli, reducing the search space.

- *Sequential and Parallel Processing.* Most HPP models employ only one type of information processing technique associated with conscious reasoning (sequential processing). However, two distinct types of information processing systems seem to exist in human cognitive processing [Rasmussen-86].

*Sequential* is the slow limited capacity processing system operating within the realm of human awareness (consciousness), while *parallel* is the distributed, high-capacity processing system operating in the sub-conscious.

Sequential processing was selected because of the fact that distributed parallel processing is highly parallel, and therefore, does not play a role in defining human performance limitations. Contrary to the rationale, distributed parallel processing does each of the following.

- Plays a role in knowledge acquisition (e.g., visual or pattern recognition).
- Fails to model parallel system, which engenders the Frame Problem.
- Accounts for the automated behavior observed in human activities that are learned.

- *Knowledge Representation Strategies.* Humans have the ability to represent and employ knowledge in different formats and at different levels of abstraction.

- *Propositionally Based Schemes.* The schemes represent relations as attribute-value pairs [Young-92] (e.g., semantic nets).

- *Analogical Representations.* These representations map characters of the external world directly onto the mind (e.g., spatial representation). As the external object changes, images undergo the same change, [Kosslyn-80, Wickens-84].

- *Procedural Schemes.* These schemes represent knowledge as procedures for employing knowledge. This "how-to" knowledge is employed in skilled behavior [Young-92].

People engaged in complex decision making formulate *fluid tasks*. They move from one conceptual level to another with relative ease. Currently, HPP models employ a one-dimensional thought process (i.e., a single line of thinking or level of abstraction is pursued) [Rasmussen-86]. This process does not conform to the current understanding of the human thought processes.

The *competence* limitations explained above have two implications on the requirements of HPP models.

- The need for Artificial Intelligence (AI), knowledge-based (symbolic), qualitative, and analogical (sub-symbolic) simulation capabilities.

- The need for multiple levels of abstraction.

The *behavioral*, or performance, requirements call for *temporal* simulation capabilities [Christensen-90].

## 1.5 The Holon Concept

The Holon concept was developed to explain hierarchical behavior of organizations, [Koestler-67, Koestler-78]. It provides a means for characterizing the dualistic behavior of hierarchically organized entities.

There are three defining characteristics of Holons.

- *A Self-Assertive Tendency.* This tendency enables holons to behave as quasi-autonomous wholes.
- *The Hierarchical Structure.* This structure controls the flow of information.
- *The Behavior.* Behavior is governed by fixed rules and flexible structures that provide the ability to represent dynamic behavior.

A hierarchical structure of Holons is called a *holarchy* (see Figure 1.1). The various levels in a holarchy are referred to as its *depth*, and represent the various levels of abstraction. The number of holons at a level determine its *span*.

Branching lines between holons represent communication channels. Holons on different levels but on the same branch communicate *only* through these channels, but holons in a different branch structure may communicate directly.

Holarchies may be classified under one of two *classes of hierarchies*.

- *Input Holarchies.* These holarchies convert complex input patterns into coded signals. For example, a motion detector picks up lines, then edges, then motion.
- *Output Holarchies.* These holarchies work inversely to the input holarchies — they elaborate or expand on current information. For example, when a door bell rings, someone is expected to be at the door, so it is opened.

Holons have the following characteristics.

- Holons have fixed rules, called *cannons* that identify their behavior.
- The cannons are applied through flexible strategies based upon current state of the holons' environments.
- Each Holon has its own world representation and control structure that determines the applicability of its cannon [flexible] (e.g., spider's hexagonal web).

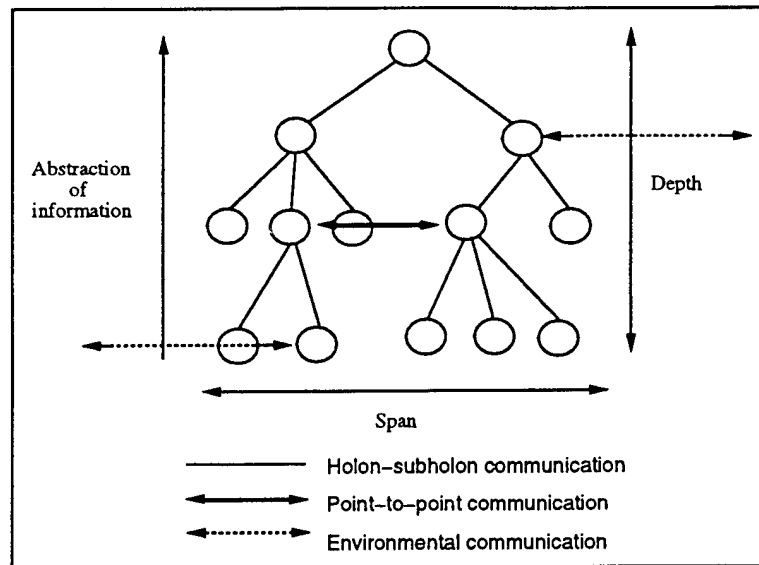


Figure 1.1: A Holarchy

- The environment for different holons within the same structure may be quite different.
- A direct correlation between the level of abstraction and the distance from the external sensor seems to exist.
- Holarchies may be modeled in n-dimensional space for varying the *complexity* and *resolution* of the model.

## 1.6 Holon Architectures for Human Performance Process Models

Holons and holarchies offer a framework to overcome the limitations of models such as Soar and ACT\* and to efficiently represent HPP models [Young-92, Young-93]. The holon architecture supports the following.

- Top-down and Bottom-up data processing.
- Multiple forms of knowledge representation through modularity and quasi-autonomous behavior.
- Three types of communication (branch, point-to-point, and broadcast) providing better representation and modeling of coordinated, concurrent behavior.
- Hierarchical structuring of cognitive processing.

- Modeling of performance limitations caused by high task demands and concurrent tasking.

## 1.7 Conclusion

Existing human performance models in general, and human performance process models in particular, only provide a partially adequate basis for developing a useful suite of models for operability analysis. Holon concepts provide a framework for constructing more adequate models. However, the holon framework needs to be instantiated with more concrete model elements in order to enable alternative analysis of computational holon architectures. A formulation of relevant intelligent system metaphors is used to suggest such elements for this framework.

## Chapter 2

# Metaphors for Human Performance Process Model Development

There are two primary motivations for creating HPP models: the long term goal of fundamental understanding of cognitive behavior of human operators and controllers of systems and the more short term goal of employing HPP models in operability analysis, (i.e, to support evaluation of operator console designs for their effectiveness). A set of modeling metaphors that suggest features that are relevant to HPP model creation are named below.

- Distributed Artificial Intelligence(DAI) [Lesser-90]
- Distributed Operating Systems(DOS) [Sha-93, Tanenbaum-92]
- Engineering Psychology(EP) [Wickens-84]
- Fallible Machine(FM) [Reason-90, Newell-90, Anderson-90]
- High Autonomy Systems(HAS)
- Intelligent Robotics(IR) [Bailey-93, Albus-90]
- Neural Darwinism(ND) [Edelman-87, Damasio-89]
- Society of Minds(SOM) [Minsky-86].

The selected metaphors are not intended to be exhaustive of all possible modeling approaches that might be relevant to HPP. Rather, they are intended to suggest dimensions for a hypothetical space in which HPP models reside, each metaphor being a kind of exemplar of a cluster within such a space. The dimensions for HPP models are listed below.

- Intelligence/competence aspired to.
- Performance attributes under consideration.
- Integration with sensing/action in the real world.
- Built-in structure.
- Emergent behavior.
- Learning mechanisms.
- Coordination mechanisms.
- Dynamic/temporal considerations.

Analyses of the metaphors along the indicated dimensions are presented in Appendix A.<sup>1</sup> The collection of dimensions and values for them identified in the metaphors provide a basis for assessing the degree to which language environments and computer platforms can support development of HPP models. The development of functionality dimensions is discussed in more detail in Chapter 3. Let us first discuss the use of metaphors in the development of HPP models.

Although it is more obvious in the case of the long term research goal, the metaphor approach serves both short and long term goals. In the long term case, a cognitive science testbed should be able to support all aspects of the metaphors to the extent that they are formulated in cognitive science terms. In the short term case, to create valid and predictive HPP models for operability analysis may necessitate bringing into the model certain features that bear upon the particular modeling objectives. In the later case, it is useful to have guidance as to which metaphors deal with objectives-relevant features and which environments/platform will be necessary to support such features. At this stage in HPP modeling and cognitive science knowledge, such guidance can be only heuristic and suggestive rather than definitive in nature.

Figure 2.1 sketches some possible paths for creating HPP models based on the metaphors. Each arrow represents a possible contribution of a metaphor. Significant obstacles block progress along these paths, as shown in Table 2.1. A brief discussion of these issues follows.

- EP suggests the framework for observation of operator behavior and performance measurement, although it needs further elaboration for cognitive issues. EP models are at the surface level, hence lack explanatory and predictive capability. Therefore, they motivate the consideration of the other metaphors.

---

<sup>1</sup>Dimensions and values were developed by a critical, but informal, examination of representative literature in the various metaphor areas.

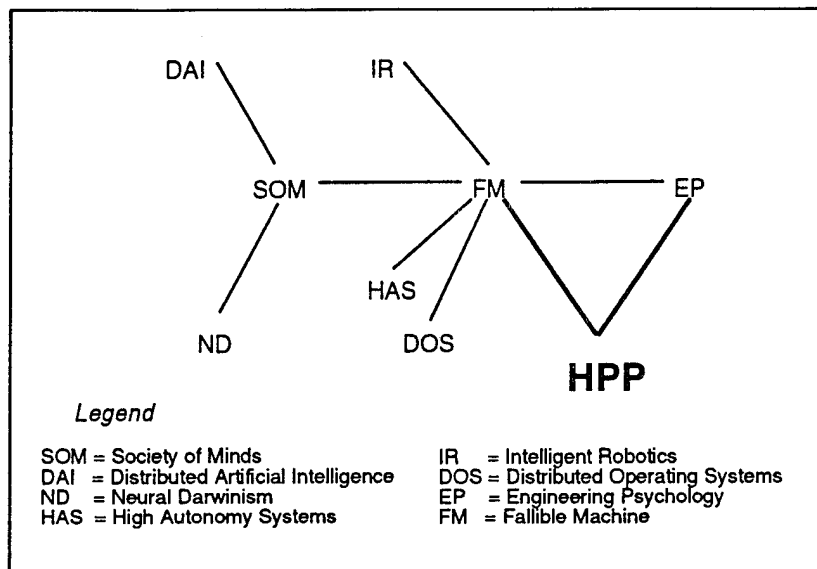


Figure 2.1: Possible Paths for Human Performance Process Model Creation

- FM is the closest in spirit to the HPP requirements since it postulates a cognitive architecture with both competence and performance features. However, FM cognition is weakly integrated with sensory input and action output. Also, since FM is stated at the symbol machine level, its performance postulates are mostly *ad hoc*.
- SOM, IR, HAS, and DOS metaphors may contribute to strengthening the FM explanatory and predictive capacity. SOM provides micro-level structures that could ultimately explain (or invalidate) FM features. Unfortunately, it is not a well-specified theory with clear predictive or explanatory constraints. IR, HAS, and DOS are engineering paradigms that could be applied to provide the FM with greater position in the real world. The three complimentary paradigms could suggest well specified sensing/action, fault recovery (i.e., error handling as opposed to FM's error generation), and resource management mechanisms, respectively. These mechanisms, however, require translation from the computer-based substrate in which they are stated to wetware-based architectural contexts.
- DAI's agent-coordinating mechanisms could fill in some of the details of agent interaction that are missing in SOM; however, its high intelligence problem-solvers are not plausible as primitive SOM agents.
- Finally, ND provides a microlevel theory that could underlie SOM postulates, or even FM postulates (if SOM is bypassed). Although comprehensive in scope, ND is a mostly sparse framework for bridging the gap between neuroscience and cognition.

An example of HPP model creation which combines metaphors is the Adaptive Decision Maker (ADM) model recently proposed by Morgan [Morgan-93]. SOM's concepts of frame

Table 2.1: Contributions of Metaphors to HPP Modeling & Barriers to Application

METAPHOR	CONTRIBUTION TO HUMAN PERFORMANCE PROCESS MODELING	BARRIER TO OVERCOME
DAI	agents and their coordination apply to SOM	agent capability sophistication too high
DOS	task/resource/time management competence/performance suggestive	technology specific
EP	performance measurement/models	macro level models
FM	close to HPP in spirit	missing link to sensing/action
HAS	hierarchical organization suggestive	technology specific
IR	sensing/action suggestive	technology specific
ND	neural->micro->macro transition performance suggestive	gaps between levels
SOM	micro->macro transition competence suggestive	sketchy, underspecified

arrays and A-brain-B-brain mechanisms are combined with control theoretic concepts underlying High Autonomy Systems and Intelligent Robotics to create a model containing Supervisory and Execution layers, each characterized by groups of frames representing repertoires of skills [Minsky-86]. Grouping into task and situation-related frames reduces work-load and response time but limits capability to recognize and react to events not stereotyped in the currently executed frames. The Supervisor layer selects and instantiates execution frames, but this action imposes a workload on the next higher cognitive layer. Thus, Morgan claims that the ADM model accounts for the failure of expert decision-making to meet strict optimality criteria, [Morgan-93]. Rather, it shows that in a time-critical environment, recognition-primed decision-making, bounded rationality, and fixation appear as natural consequences of measures taken to avoid being overwhelmed by situational complexity. While not explicitly present, further elaboration of the ADM model would clearly relate to elements of the DOS metaphor, since frames can run concurrently and must be switched in and out of working memory (to use a Fallible Machine construct).

## Chapter 3

# Human Performance Process Functionality Requirements

From the the analyses of metaphors in the appendix, sets of elements associated with each dimension have been compiled. These sets provide a fairly comprehensive enumeration of the functionalities that may be required in HPP model construction. Note that further division into neural, symbolic, and functional levels are employed in some cases to help organize the elements. The following sets coarsely characterize the representation level of the metaphor from which the element was extracted.

- **Intelligence/Competence.**

- Survival in real world.
- Higher-order thinking, problem-solving.
- Collective problem-solving.
- Common knowledge question answering.
- Autonomous, adaptive, fault-tolerant plant operation.
- Visual navigation of semi-structured environment and manipulation of objects in it.
- Guaranteed forward progress (non real-time) or gracefully degrading (real-time).
- Task execution.

- **Performance Attribute.**

- Dimensions: efficiency, bias, optimality.
- Reaction Time, latency, accuracy, speed accuracy tradeoff.
- Information-transmission Rate.
- Errors: misperception, mode, slips.
- Effects: mental workload difficulty, composition, stress, individual differences.

- Throughput, job completion time, utilization of resources.
- Meet cost, resource, time bounds.
- Diseases of consciousness.
- **Sensing/Action in the real world.**
  - Sensors: sensor fusion and interpretation.
  - Effector: output elaboration hierarchy.
  - Continuous, tightly coupled integration.
- **Built-in Structure.**

*Primitives:*

1. Neural.
  - Neuronal groups.
  - Reentrant signaling.
  - Hedonistic value system.
2. Symbolic.
  - Agents (rudimentary).
  - Specialized connector-agents.
  - Frame-arrays.
3. Functional.
  - Agents (high level problem solvers).
  - Peripheral working memory.
  - Focal working memory.
  - Buffer store.
  - Knowledge base (procedural and declarative).
  - Hierarchical problem space decomposition capability.
  - Production rule engine.
  - Shared communication medium, blackboard.
  - Value assigner.
  - Goal search mechanisms.
  - Maps/spatial representations.
  - State-described processes.
  - Models of external world and own components.
  - Synchronization mechanisms.
  - Process state saving/restoration.
  - Scheduler.
  - Processing elements: stages, modalities, codes, responses.

- Cognitive resources.

*Architecture:*

1. Neural.
  - Neuronal group properties and reentrant connectivity.
  - Neuroanatomical structures and pathways.
  - Maps (sensor receptor sheets to nervous system regions).
2. Symbolic.
  - Agencies.
  - Heterogeneous, loose coupling of agencies.
3. Functional.
  - Knowledge-based parallel retrieval.
  - Interprocess communication.
  - Trajectory/path planning.
  - Task scheduling hierarchies.
  - Task execution hierarchies.
  - Fault management hierarchies.
  - Resource allocation.
  - Processing and memory capacity constraints.

• **Emergent Behavior.**

1. Neural.
  - Categorization.
  - Memory as recategorization.
  - Primary and secondary consciousness.
2. Symbolic.
  - Visual ambiguity.
  - Language understanding.
  - Serial, distributed processing.
3. Functional.
  - Laws: Fitts, Hick-Hyman.
  - Automatization levels: skill-based, rule-based, knowledge-based.
  - Attention (selective, focused, divided), time-sharing.
  - Psychological Refractory Period (PRP).
  - Processing: serial, parallel.
  - Error-prone question answering.
  - Wandering attention.
  - Cooperation to solve problems.
  - Reactive recovery from anomalies and malfunctions.

- Collision avoidance.
- Movement/manipulation goal attainment.

- **Learning Mechanisms.**

1. Neural.
  - Neuronal group selection.
  - Neural net back propagation, etc.
2. Symbolic.
  - Accumulating.
  - Uniframing.
  - Reformulating.
  - Trans-framing.
3. Functional.
  - Chunking of knowledge elements into units.
  - Case-based reasoning.
  - Modeling.

- **Coordination Mechanisms.**

1. Neural.
  - Resolution of conflicts.
  - Cross-modal construction.
  - Recursive synthesis.
2. Symbolic.
  - Principle of noncompromise.
  - Interruptability, recursive (stack-like) memory.
  - Suppressor-agents, censor-agents.
  - Exploitation of agents using model.
3. Functional.
  - Distributed, Localized: information-sharing, negotiation, contracting, delegation of authority, abstraction constrained (top-down/bottom-up) processing.
  - Hierarchical control.
  - Synchronization, prioritization, load balancing, flow control.

- **Dynamic/Temporal Considerations.**

1. Neural.
  - Stability characteristics.
  - Signaling patterns.
  - Race competitions.

/newpage

2. Symbolic.

- Time blinking.

3. Functional.

- Transfer functions, time domain, frequency domain.
- Crossover model, optimal control model.
- Serial Reaction Time (RT) paradigm.
- Time-sampling, slicing.
- Transient/steady state performance.
- Time horizons.
- Motion of effectors and objects in space.

### Summary

A comprehensive enumeration of the functional dimensions of intelligent system models was just given. This listing serves as a "menu" of objectives that HPP models can be built to satisfy. In Chapter 4, the ability of software environments to support the enumerated functionalities is examined. However, first is an explanation of the perspective on system requirements.

## Chapter 4

# General Requirements for Languages/Environments

Although concepts for integrated modeling and simulation support systems have been around for some time [Oren-79], only recently have serious efforts been made to implement such concepts. Henriksen [Henriksen-83] recognized the need for support for the model development process in addition to the runtime execution facilities provided by most early simulation languages. Balci [Balci-86] stated requirements for supporting model development in analogy to software development paradigms. Balmer [Balmer-86] and Murray et al. [Murray-87] proposed employing the newly emerging tools of AI and knowledge-based systems to support aspects of the simulation enterprise. Reddy et.al. [Reddy-86] departed from traditional procedure-based simulation languages to incorporate declarative knowledge representation and reasoning facilities. Klahr [Klahr-86] revived interest in the object-oriented (OO) basis of simulation modeling concepts. The genesis of OO concepts is to be found, not in Smalltalk [Goldberg-83], but in Simula, a simulation language developed as early as 1965, [Zeigler-91]. Ruiz-Mier and Talavage [Ruiz-89] addressed the requirement to integrate both declarative and procedural knowledge-representation schemes for models of intelligent agents. Evidence now supports that knowledge-based simulation can serve as a strong foundation for complex systems design [Reason-90]. Long-term simulation model engineering and evolution are considered by Fishwick [Fishwick-90] and the fundamental requirements for standardized simulation environments by Tanir and Sevinc [Tanir-93].

The following were identified as dimensions for requirements of languages/environments for HPP modeling and simulation.

- Specific HPP-Required Functionalities.
- Software Development Support.
- Model-Data Base Management.
- Simulation Model Specification/Execution.
- Support for Model-to-Architecture Mapping.

- Support for Multiple Resolution Levels.

The specific HPP-Required functionalities have already been discussed in the previous chapter. The next section provides a breakup of each of the dimensions into more specific elements.

## **4.1 Requirements for Language/Environment Support of Human Performance Process Modeling**

The following are requirements for language/environment support of HPP modeling.

### **1. Software Development Support.**

- Enhanced model quality integral through development process.
- Comfortable prototyping/development/reification.
- Stand-alone modules.
- System composition from components.
- Top down design/bottom up testability.
- Configuration management.
- Visual interface access to environment.
- Output analysis, visualization, animation.

### **2. Model-Data Base Management.**

- Integrate with standard database management systems.
- Support hierarchical synthesis process and model object hierarchies.
- Exploit model components resident in a model object base.
- Support organization and reuse of objects and models.

### **3. Model Specification/Execution Simulation.**

- Support continuous, discrete event and related dynamic system formalisms.
- Support AI knowledge representation formalisms: goal planning, rules, frames.
- Support multiple concurrent agents.
- Support model evolution through hierarchical, modular construction.
- Stable real-time simulation.

### **4. Model-to-Architecture Mapping Support.**

- Support mappings of models onto diverse computer architectures.
- Facilitate upgrading to higher capability platforms through software compatibility.

- Assure correct simulation process.
- Generate alternative architectures and mappings.
- Experiment with/select efficient (time/space) alternatives.

#### 5. Support for Multiple Resolution Levels.

- Refine models to provide high fidelity representation.
- Coarsen models to meet execution time and space constraints.
- Integrate into the model base via abstraction mechanisms with explicit links for state and parameter data bases.
- Cross-validate models w.r.t. other (high or low resolution) models.

## 4.2 Candidate Languages/Environments for Human Performance Process Model Development

Having developed requirements for support of HPP model development, candidate languages and environments that might meet these requirements are now considered. Such software systems can be broken down in general and special purpose categories, with the latter further divided into simulation, artificial intelligence, and cognitive architecture support. None of the existing systems can satisfy all of the requirements that have been identified. The reasons for this conclusion are discussed below.

- *General Purpose Languages/Environments.* General purpose languages most commonly employed today are the C programming language, for numerically-based science and engineering problems, and Common Lisp for AI symbolic manipulation work. Object-oriented extensions, C++ and Common Lisp Object System (CLOS) enhance the power of the respective basic languages considerably. Although they provide strong foundations for software development, such systems do not provide the primitives that are available in the special purpose languages below.
- *Special Purpose Languages/Environments.* The two types of special purpose languages/environments are as follow.
  - **Simulation.** A wide variety of commercial simulation languages can be found on the market, ranging from highly specialized program generators (e.g., SIMFACTORY, SIMNET) to languages providing general discrete event modeling facilities (e.g., GPSS, SIMSCRIPT, SIMAN)<sup>1</sup>. Although they provide primitives for expressing discrete event dynamics, such languages do not provide the particular facilities required for HPP model development.

---

<sup>1</sup>For a detailed exposition, see Software/Modelware Tutorials, Winter Simulation Conference Proceedings, 1991.

- **Artificial Intelligence.** Originally developed as extensions of general purpose Lisp dialects, environments for expert system development had a short-lived commercial interest in the 1980s. Such expert system shells<sup>2</sup> are usually too restricted in their knowledge representation schemes to serve as a basis for HPP modeling. However, two environments, ART-Enterprise and ProKAPPA, contain comprehensive facilities for knowledge representation and reasoning, potentially offering full-fledged support of HPP model development. They are briefly described in Appendix B and evaluated against HPP requirements in Table 5.1. Summarizing this evaluation, these environments are strong in knowledge representation and reasoning employed in artificial systems. However, they are not oriented to express the mechanisms characteristic of natural cognitive systems, particularly such **efficiency reducing** features as fallibility, being accident prone, memory recall limitations, and so forth. Also evaluated is Simkit, which combines some discrete event simulation features with Knowledge Engineering Environment (KEE) and which is an ancestor of ProKAPPA. This package is no longer supported. Had it been developed further, it might have combined sufficient simulation and AI primitives to provide a serious contender for consideration.

Also falling within the AI category are neural networks (reviewed in Appendix C). In principle, these software packages could provide a basis for HPP models developed at the neural level. However, the artificial neurons employed typically bear only the faintest resemblance to counterparts found in real brains. As with expert system shells, whether the constraints found in application-oriented packages can be overcome to serve as a basis for HPP model development is questionable.

- *Object-Oriented Knowledge-Based Simulation Environments.* As suggested in the introduction to this chapter, environments based on object-oriented, knowledge-based foundations come closest to meeting the requirements for supporting HPP model development. Two such environments are considered: SCORE/SPROCKET [Palmucci-92], developed at BBN as the platform for OMAR, is described in Appendix B; Discrete Event Specification (DEVS-Scheme) [Christensen-90], based on a generic modeling formalism, is described in Appendix D.3. These environments provide a good starting point for the development of an HPP modeling environment.

---

<sup>2</sup>See PC AI, Jul-Aug, 1993 for a complete listing.

## Chapter 5

# Evaluation of Languages/Environments

Languages/environments are evaluated with two respects.

1. With respect to the specific functionalities required for HPP modeling.
2. With respect to the support of model development dimensions enumerated earlier.

Table 5.1 shows the evaluation of some of the most capable, commercially available AI languages/environments with respect to HPP functionality requirements. Each entry represents an estimate of the time that would be required to synthesize the functionalities listed in the first column (referring to the breakup in section 4.1). The scale is an arbitrary 0 to 10, where 0 means this functionality is available directly, and 10 indicates it would have to be developed from scratch. The scale is therefore a measure of the distance from the functionality to the primitives of the environment given its construction tools. A value = x, indicates this functionality would be impossible with the given language (Caveat: Since each of the environments includes universal computation facilities, the effort required would be inordinate).

Table 5.2 presents the evaluation of SCORE/SPROCKET and DEVS-Scheme simulation environments with respect to model development support. Also included is an overall assessment of HPP functionality capabilities. Note that SimKit is included in both tables to provide a common reference for comparison.

ART-Enterprise and ProKAPPA, although powerful for built-in symbolic knowledge representation (evaluation = 3), generally do not support the *efficiency reducing* characteristics of interest in HPP models mentioned earlier. Moreover, they do not address other functionality categories such as performance, sensing/action, and so forth.

The Dynamic/Temporal features needed for HPP models must be built up from scratch in the AI shells. Table 5.2 shows that these features are naturally supported by the simulation domain (see Simulation Model Specification/Execution). Moreover, since the simulation

Table 5.1: Evaluation of AI Languages/Environments with respect to HPP Functionality

	<i>ART</i>	<i>ProKAPPA</i>	<i>SimKit</i>
<b>Performance attributes:</b>	10	10	5
<b>Sensing/Action in the real world:</b>	10	10	10
<b>Built-in Structure:</b>			
<i>Primitives:</i>			
neural	x	x	x
symbolic	3	3	3
functional	9	7	5
<i>Architecture:</i>			
neural	x	x	x
symbolic	5	5	5
functional	7	7	9
<b>Learning mechanisms:</b>			
neural	x	x	x
symbolic	10	10	10
functional	5	10	10
<b>Coordination Mechanisms:</b>			
neural	x	x	x
symbolic	10	10	10
functional	10	10	10
<b>Dynamic/Temporal Considerations:</b>			
neural	x	x	x
symbolic	x	x	5
functional	10	10	5

Table 5.2: Evaluation of Simulation Environments with respect to Model Development

	<i>SCORE</i>	<i>DEVS-Scheme</i>	<i>SimKit</i>
Software Development Support	4	5	4
Model-Data Base Management	8	2	5
Simulation Model Specification/Execution	3	4	7
Model-to-Architecture Mapping	10	5	10
Multiple Resolution Levels	10	5	10
HPP-Functionalities	5	6	8

Time to synthesize (0 = none, 10 = start from scratch, x = cannot be done).

environments are based on powerful object-oriented foundations, they also can be developed in the direction of HPP functionalities, as can the AI environments.

#### Conclusion:

Overall, the best approach to modeling environments is to “grow” them from simulation rather than AI foundations.

## Chapter 6

# Review of Parallel Processing Concepts

To set the groundwork for HPP hardware support discussions, reviewing some vocabulary pertaining to parallel processing architectures and computations. There is considerable variation in meanings of terms within the architecture community, so a certain degree of multiplicity of reference must be tolerated. The review is presented in the form of two sets of dimensions, one for architectures and one relating to problems (algorithms, computations). The problem-related dimensions correspond closely to the architecture ones. Indeed, they relate to problem decompositions — that is, once a problem is decomposed into subtasks that can be assigned to processors, it is relatively straightforward to match its characteristics to those of a hardware architecture. At this stage in the evolution of parallel processing, more abstract and independent characterizations of which problems run best on which architectures are not available.

### 6.1 Dimensions of Architecture

An effort is made to provide a working definition of architectural dimensions to prevent confusion in further discussion of these concepts.

- *Flynn Class.* A computer system can be viewed in terms of streams of instruction (I) and data (D), each of which can be single (S) or multiple (M). SISD characterizes the conventional serial Von Neumann architecture. SIMD refers to architectures in which a single control processor coordinates the computation of multiple processors, typically in synchronous (lock step) fashion. MIMD denotes architectures in which, typically, multiple equally-autonomous processors cooperate in asynchronous fashion. MISD has very limited application.
- *Processor Granularity* (fine, medium, coarse). Process Granularity refers to the word size (bits) and computing power (instructions/sec or floating point operations/sec) of processors.

- *Interconnect Topology* (bus, hypercube, mesh, tree, crossbar, Multi-stage Interconnection Network (MIN) etc.). The graphical structure formed by the communication links transmitting control and data among processors is the interconnect topology. This structure determines which processor pairs can communicate directly and which must communicate indirectly through multiple hops. Of course, the capacity of the links to transmit data (bandwidth measured in bits/sec, latency in seconds) is also critical (e.g., “fat” trees concentrate capacity near the root (where traffic is heavy) rather than at the leaves) <sup>1</sup>.
- *Memory Distribution* (Shared/Distributed). Processors may share a single memory space or have private memories and share data via message passing communication. Shared memory may be a virtual construct presented to the programmer where, in fact, each processor has a local cache, but the system automatically maintains consistency of data copies in multiple caches through established cache coherence protocols [Hwang-84].

## 6.2 Dimensions of Problem Decomposition

The features of applications that make them suited or unsuited for massively parallel processing (inspired by Furtney and Taylor [Furtney-93] but considerably different). As indicated earlier, the dimensions actually refer to problem decompositions, of which there maybe many for a given problem. The following are some of the dimensions that may be considered for problem decomposition.

- *Parallelism* (small, medium, massive). The number of processors that can be kept busy during the computation determines the parallelism.
- *Regularity of Local Computation/Communication Interaction* (low, high). Sub-tasks must be coordinated to achieve the overall task; thus, sub-task execution alternates with sending/receiving of data and control. The complexity of the processor/network interface (e.g., synchronization, interrupt handling) is lower when such alternation is regular (predictable in advance) than when highly irregular.
- *Regularity of Information Neighborhood Structure* (low, high): The information neighbors of a sub-task are those sub-tasks with which it needs to exchange information. Decompositions with regular neighborhoods can better conform to a matching interconnect topology and/or impose less traffic load on multistage complete interconnects.

Massively parallel SIMD architectures support problem decompositions with massive parallelism, highly regular computation/communication interaction, and information neighborhood structure. MIMD architectures with specialized network interface hardware better

---

<sup>1</sup>From a theoretical standpoint, fat trees can be shown to be a nearly universal network scheme; a fat trees routing network of any given size is always nearly the best possible network of that particular size. Other networks described in the literature do not demonstrate the combination of universality, locality, bandwidth variability, or load balancing characteristics of the fat trees [Ramanathan-93].

support irregular computation/communication interaction. High capacity/high connectivity interconnects are needed to cope with irregular information neighborhood structures.

These ideas are illustrated in the Numerical Aerodynamics Simulation (NAS) parallel benchmarks which focus on eight important aspects of highly parallel supercomputing for aerophysics applications. These applications include the types of actual data movement and computation required in parallel processing [Bailey-93].

The Embarassingly Parallel Benchmark is typical of many Monte Carlo applications. Because it requires little or no communication, this kernel benchmark estimates the upper limits of a system's floating-point performance (see Figure 6.1).

Figure 6.2 show how three types of NAS benchmarks degrade the performance of the Connection Machine-2 (CM-2). The simplified Multigrid benchmark solves a 3-D Poisson partial differential equation. This code is a good test of both *short-and-long-distance communication*, although the communication patterns are *highly structured*. The Conjugate Gradient benchmark uses a method to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric, positive, definite matrix. This problem is a typical unstructured grid computation in that it tests *irregular long-distance communication*. The Integer Sort tests both integer computation speed and communication performance. This problem is unique in that floating-point arithmetic is not involved, although significant data communication is required. It is the benchmark most relevant to symbolic processing requirements.

### 6.3 Examples of Problem/Architecture Pairings

In principle, all decompositions of a problem must be investigated to assess its amenability to parallel implementation. However, often the types of decomposition under consideration are highly constrained (e.g., by the programming language employed). Under such constraints, it makes sense to equate the characteristics of a problem with those of its decompositions. The following examples may help clarify this point.

#### 6.3.1 Data Parallel Model

Data parallel computation was popularized by the Connection Machine, first in its early SIMD versions (CM-1,2,200) and recently in its MIMD style, CM-5 (see Appendix G for synopsis). In the *data* as opposed to *control* parallel model, homogeneous local operations on a large number of data items, executed in lock step alternate with global aggregation and redistribution of results. Algorithms expressed in the data parallel model (e.g., programmed in CM-specific code) can achieve very high performance on architectures geared to support the requisite local computation and global communication for a large number of processing elements (millions of processors).

### 6.3.2 Artificial Neural Nets

Artificial neural nets, as opposed to biological ones, are characterized by large numbers of quasi-neurons with simple numeric processing, typically with layered interconnection patterns, transmitting simple numeric data (see Appendix C for synopsis). One form of network decomposition maps each layer to a processor in a coarse grained MIMD architecture. Alternatively, layers can be pipelined with each stage mapping neurons to processors with a one-to-one correspondence [Adaptive-93].

Artificial neural nets (ANN) are of interest as first approximations to sub-symbolic level models such as ND and SOM. More realistic neural nets are characterized by greater neuron functionality and irregularity of both computation/communication interaction and information neighborhood structure. Computer vision architecture studies offer some clues as to which kinds of computer architectures might support such processing [Arbib-89].

## 6.4 Heterogeneity of Processing in Performance Process Models

Figure 6.3 schematizes the basic functionalities involved in HPP models and associated processing styles. The scheme suggests the need for heterogeneity (i.e., the application of a multiplicity of processing styles) to support HPP simulation [Khokhar-93]. The evidence for the sensory processing component is derived from the design of computer-image-understanding architectures [Shu-90]. Three levels of vision processing are distinguishable: low, intermediate and high.

Low level, or early, processing converts an input image to an output image of approximately the same size employing operations such as edge detection, corner extraction and connected component labeling. This level is suited to fine grained SIMD processing, since operations are synchronously applicable to all pixels and have regular neighborhood structures (such as immediate geometrical neighbors).

Intermediate processing, interfaces between low and high, involves operations such as grouping of corners to extract candidate rectangles, which map the input image to a smaller set of scene-related features. Medium-grained MIMD processing is best suited at this level because operations can be computationally intensive and involve non-local, irregular information-flow neighborhoods.

High level processing, or image understanding, involves inferring semantic attributes from the candidate features supplied by lower levels (e.g., confirming the existence of rectangles matching scene elements in a knowledge-base). These algorithms employ complex symbolic data structures, are non-deterministic (irregular computation/communication interaction), and appear to require coarse-grained MIMD processing.

Simulation studies on the Defense Advanced Research Project Agency (DARPA) Integrated Image Understanding Benchmark suggest that heterogeneous image understanding architectures [Shu-90] performed better than any single machine considered [Khokhar-93].

The associations between functionality and processing style in Figure 6.3 are tentative extrapolations from such evidence. First, computer vision is generalized to human vision processing, which is then generalized to all forms of sensory processing. Second, motor output generation is assumed to involve the "inverse" of sensory processing in performing successive elaborations of a small set of high level commands (in accord with input and output holarchy concepts). Third, overall conscious control is assumed to be serial in nature and very concentrated in a SISD, while automatized cognition hypothetically involves stratified parallel processing similar to the vision hierarchy.

Since each of these extrapolations is questionable, the suggested associations are best regarded as initial working hypotheses. The point, however, is that heterogeneity in processing requirements is highly likely for HPP simulation support.

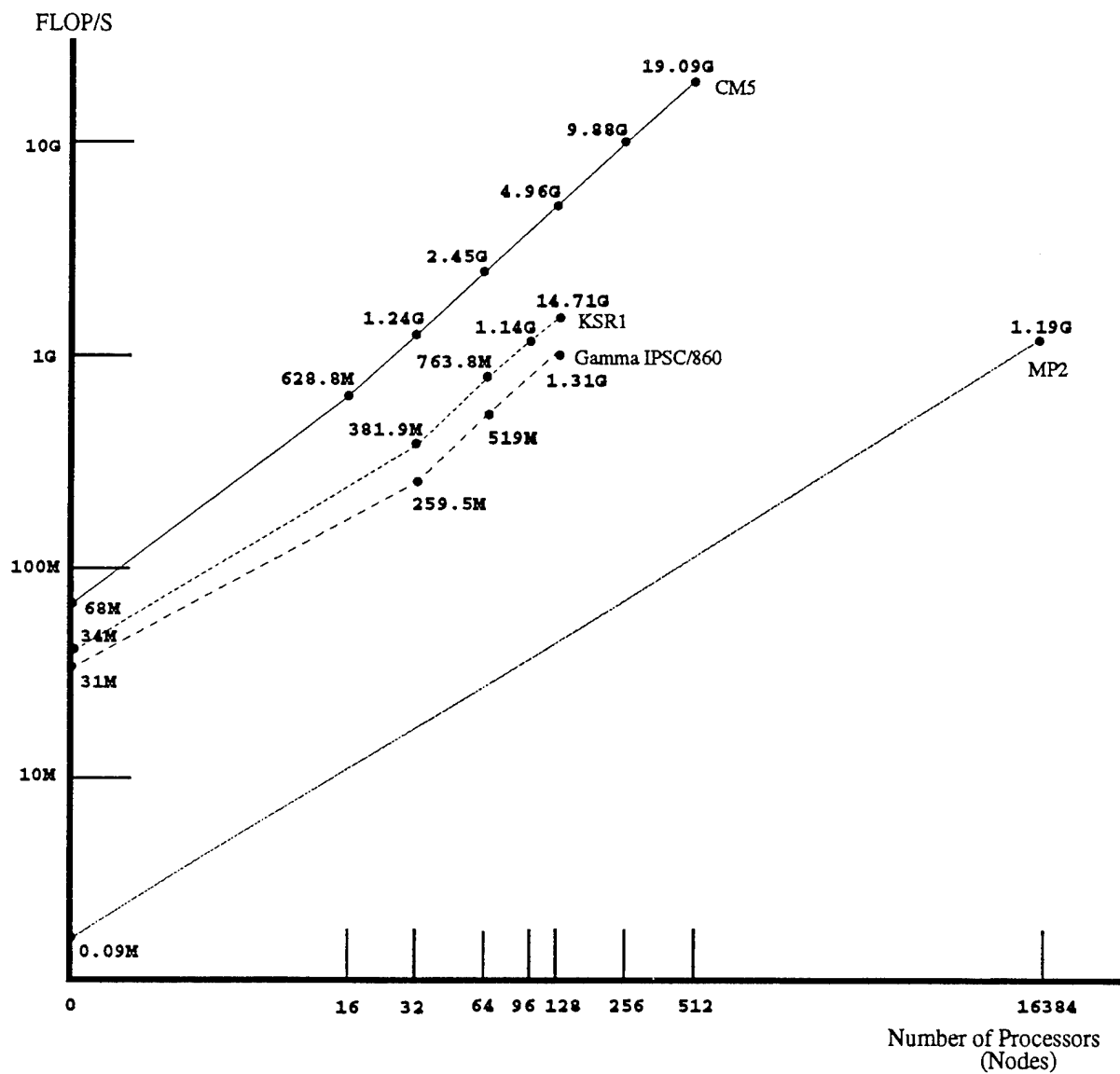


Figure 6.1: Numerical Aerodynamics Simulation Results for Embarassingly Parallel Benchmarks

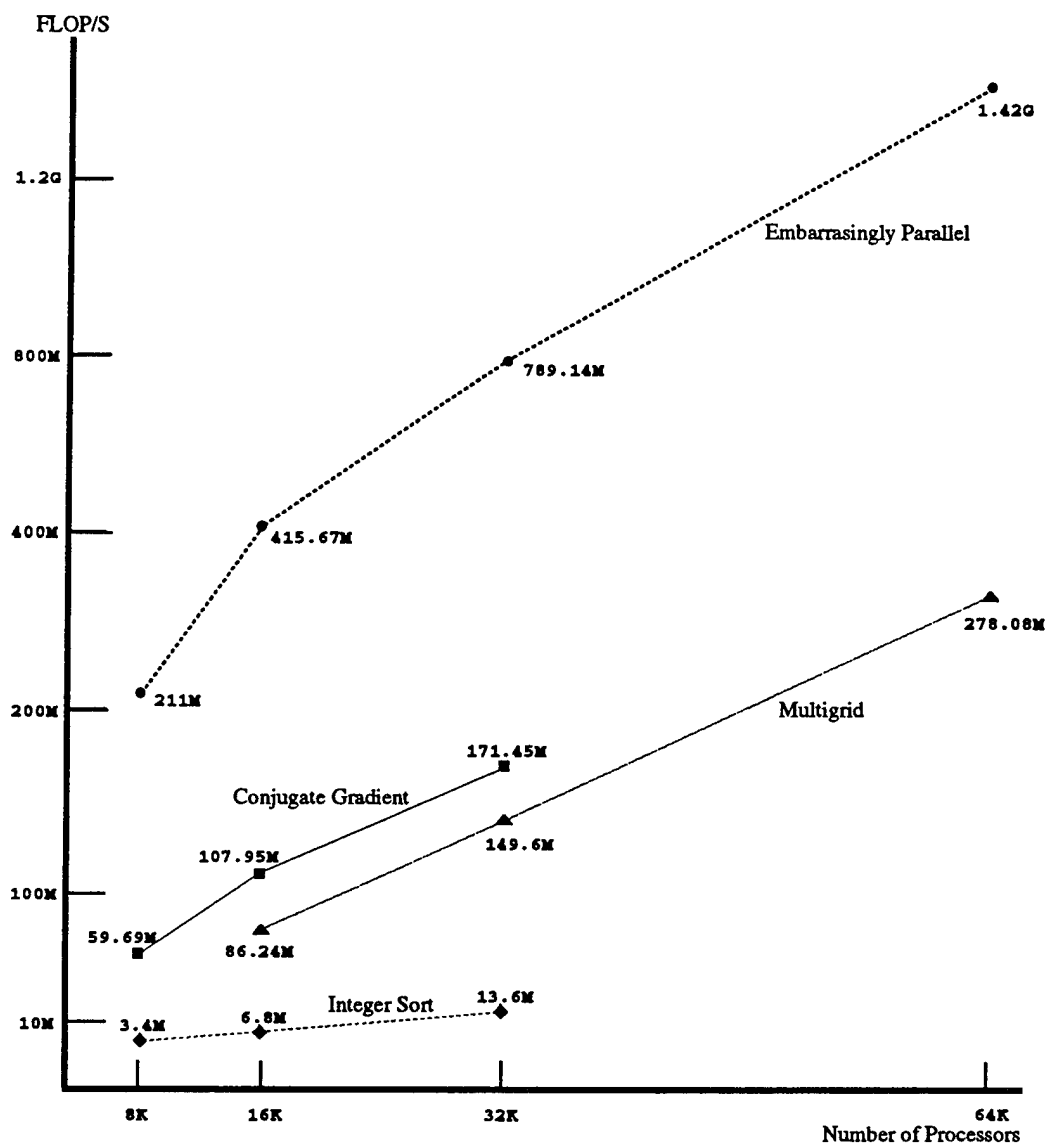


Figure 6.2: CM2: Numerical Aerodynamics Simulation Benchmark Results

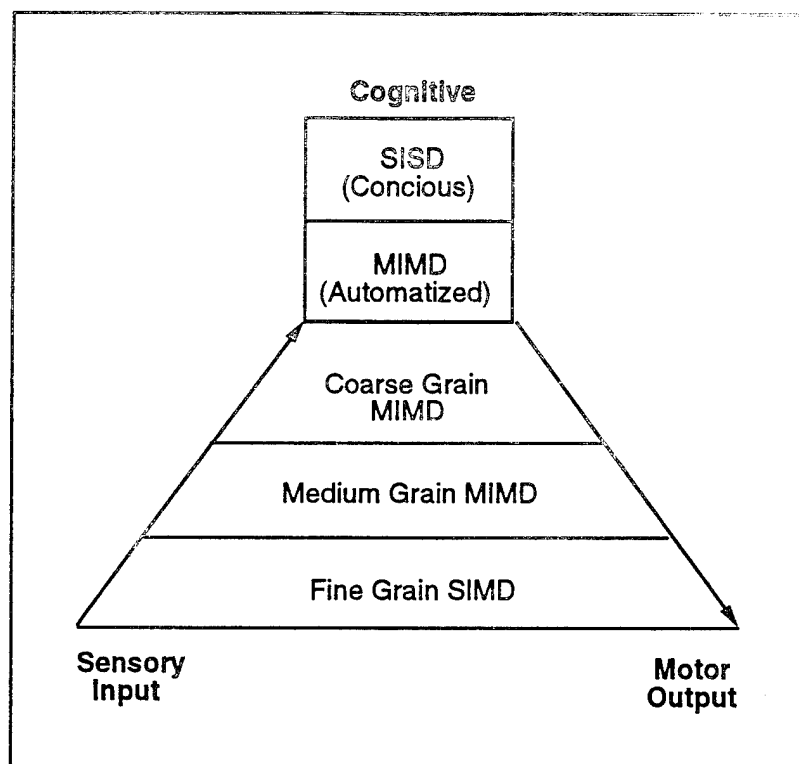


Figure 6.3: Human Performance Process and Architectures

## Chapter 7

# Hardware Architecture Requirements

The suggested requirements for hardware platform support of HPP modeling and simulation are as follow.

- *Heterogeneous Processing Support.* As discussed earlier, HPP modeling and simulation is likely to require a mix of processing styles.
- *Real-time Simulation Support.* Human-in-the-loop applications will require that the HPP simulation emulate both the nature and the timing of human response. Real-time requirements impose additional constraints on a platform since bounds are placed not only on average, but also on individual processing times. Nonreal-time simulation will also be an important task for HPP support. Nonetheless, consideration of real-time requirements is useful, since imposing such additional constraints does the following.
  - Tends to significantly narrow the alternatives, thus lending coherence to the considerations.
  - Provides an upper bound on capability, in the sense that if real-time operations can be achieved, so can nonreal-time operation.
  - Provides a useful starting point from which to relax constraints.

Real-time considerations are discussed in greater detail in Appendix D.1.

- *Evolvability from Single Workstation.* Since OMAR is being developed on a single workstation, it would most natural to evolve it from that platform to higher performance platforms as deemed necessary or desirable. The following are aspects of such evolvability.
  - *Upgradable Software Compatibility.* The software developed on a platform should continue to run on, or be conveniently adaptable to, successor incarnations of the platform.

- *Scalability.* The platform should readily accommodate additional resources. Particularly, in the context of parallel platforms, scalability refers to the ability to add additional processors to the original system maintaining linear performance enhancement. The idea is that the system performance should scale upward with increasing problem size and matching resources.
- *Hierarchical Expandability.* The platform should accommodate growth in the model complexity, a more demanding version of scalability. This concept is discussed in greater detail in Chapter 10 .

## Chapter 8

# Hardware Architectures

In view of the requirements set forth in the previous section, we consider four levels of computing platforms: single workstations, homogeneous networks of workstations, high performance machines, and heterogeneous computing environments. This section briefly reviews the salient architectural features of each of these categories. Synopses of representative platforms are presented in an Appendix F.

### 8.1 Single Workstations

A workstation offers the convenience of supporting all surveyed software and, of course, is the chosen platform for OMAR development. Moreover, the on-going exponential advances in VonNeuman-based computing power suggests that single workstations be considered as viable candidates for meeting HPP requirements for some time in the future.

### 8.2 Networks of Workstations

Local Area Networks (LANs) connecting workstations with Ethernet or other network protocols are commonly employed for multi-user distributed computing. However, such networks can serve as well as computing platforms dedicated to parallel/distributed processing applications such as HPP modeling and simulation. Currently, the interconnection networks for loosely coupled workstation clusters have long latencies and low communication bandwidths in comparison with tightly coupled supercomputers. However, low entry cost, evolvability from single workstation platforms, and recent developments in hardware and software suggest that networked workstations are very viable platform alternatives.

Hardware/software support for networked workstations that offers great promise for HPP application are briefly described below.

- *Distributed Operating System.* The distributed operating system provides low level services to facilitate the management of hardware resources (Central Processing Unit,

Input-Output, and memory) and support for high-level resource abstractions used in the building of application programs. Manipulation of system resources is enabled by a small set of machine-independent abstractions through the integration of memory management (files) and communication functions (sockets). As a result, a distributed operating system has the following goals.

- Distribution of jobs to machines for efficient use of resources.
- Transparency to make the collection of machines act like a single system.

Recently, distributed operating systems have achieved major improvements such as UNIX upward compatibility, improved distributed file management performance, real-time support, and fault-tolerance.

- *Realtime Operating System.* A computation is regarded to be *real-time* if, and only if, it has a prescribed completion-time constraint representing its urgency (i.e. time critically) as one of its acceptability criteria. Therefore, an operating system is real-time to the degree that it statically or dynamically manages resources with the objective of enabling application computations to meet their completion-time constraints. Real-Time computing systems are critical in such applications as telecommunication systems, defense systems, aircraft, and space stations.

A Real-Time Operating System (RTOS) must include the following basic features: task management, memory management, local shared memory, local semaphores, timer access, and hardware independence. *Task management* includes actions such as creating, destroying, blocking, waking up, setting priorities, and scheduling of concurrent tasks. *Memory management* is the allocation and deallocation of physical memory. *Local Shared Memory* allows tasks on the same processing unit. *Local semaphores* provide basic synchronization for tasks on the same processing unit. *Timer access* allows the execution of tasks to be controlled by time. *Hardware independence* is a virtual machine layer, which allows user software to use the hardware without having to program hardware specific code.

One of the primary concerns in a real-time operating system is predictability. In theory, the rate monotonic algorithm is used to ensure predictable execution. The response time attributes of the real-time system must fulfill a common set of goals, including predictably fast response to urgent events, high degree of schedulability, and stability under transient overload. Both the hardware architecture and the operating system should provide certain scheduling facilities for a development of practical systems.

Most distributed real-time computing systems are loosely coupled via I/O communication without directly shared primary memory. The communication facility includes links, buses, rings, switching networks, and so forth. This coupling generally results in variable communication latencies which are long in respect to local memory access times. Typical nonreal-time distributed computing systems fit the workstation model, each having an autonomous user executing unrelated local applications with statically hierarchical two-party inter-relationships. In contrast, a real-time distributed

computer system is mission-oriented, which is likely to require the cooperative execution of applications across its nodes with peer-structured inter-relationships at the application and OS layers.

- *Higher-Level Message Passing.* A set of software services that allow tools and tasks to be tied together into an open, integrated, object-oriented environment is called higher-level message passing. These services can be separated into three distinct yet related categories: message passing, subprocess control, and remote data access. Each of these services can be used independently, yet they work best in concert to facilitate tool communication in a client/server architecture. In an object-oriented environment, each stand-alone application is encapsulated, or plugged in, without source code modification and local or remote execution via the subprocess control service. The remote data access feature allows client programs to reference remote data via standard network protocols, providing a consistent mechanism to handle references to remote file system data.
- *Language-Level Extender.* The Language-Level Extender turns the networks of workstations into a parallel processing platform. Existing FORTRAN and C application codes are extended with a set of language-level primitives. The primitives enable an application program to perform optimized operations necessary for parallel processing such as spawning multiple processes, collecting results from processors across the network, storing data and managing communication among processors. Running multiple processes per workstation improves performance via better load balancing and fuller CPU utilization. The commercially available extender standards include Linda, BLAS, Parallel Virtual Machine (PVM) and POSIX. The computing power measured for Linda shows linear speedup for up to 16 workstations, the number being limited by communication bottlenecks.
- *High Speed Interconnects and Network Interfaces.* To compete with tightly coupled supercomputers, workstation clusters will require high speed and bandwidth interconnect hardware, such as Futurebus and Scalable Coherent Interface (SCI) standards [Rajkumar-89, Jhonson-93].

### 8.3 High Performance Supercomputers

*Supercomputers* are characterized as *current*, state-of-the-art, high- performance parallel machines. Supercomputers usually incorporate high computation speeds and fast and large main and secondary memories with extensive use of parallel structures and software. They involve a considerable monetary commitment for setup and maintenance.

A number of issues limit the potential of supercomputers for HPP applications.

- *High cost of procurement and maintenance* make these systems beyond the financial means of low budget HPP developmental research.

- *Specialized computing power* is one of the features that characterizes supercomputers. In their quest for speed, they seem to have lost some of the 'generalized' computing power found in workstations, and so forth.
- *Systemware* for supercomputer platforms typically support only flat architectures, making hierarchical expandability and software scalability virtually impossible.
- *Limited scalability* is an important factor that is not adequately addressed in most supercomputer systems. The systems are scalable internally (to a point) and limited in heterogeneous environments.

Synopses of representative state-of-the-art technologies are provided in Appendix G.

## 8.4 Heterogeneous Computing Environments

Advanced architectures generally only achieve a small fraction of their peak performance on many portions of real applications sets [Freund-93]. Hence, using one platform for a variety of problems is a risk that most time may be spent on tasks for which it is ill-suited. Heterogeneous computing (HC) is the well-orchestrated, coordinated, and effective use of a suite of diverse high-performance machines for high-speed solutions of problems with diverse computing needs [Khokhar-93].

An HC environment contains the following.

- *A suite of heterogeneous machines.* SIMD, MIMD, special purpose platforms such as C-NAPS for neural nets [Adaptive-93] and workstations are heterogeneous machines (see Figure 8.1).
- *An intelligent high-speed interconnection network.* Bandwidths of the order of 1 Giga-bit/sec are needed to transmit problems among machines. The network interface protocol should be off-loaded to special purpose hardware to reduce the computation/communication burden on machines. The High Performance Parallel Interface (Hippi) networking protocol standard is being developed for this standard [Arnould-91]. Intelligence is required to dynamically allocate sub-tasks to machines based on suitability and availability.
- *A user-friendly programming environment.* Two approaches, user-directed and compiler-directed, can be distinguished for partitioning of problems into tasks and their dynamic allocation to machines.

Several advantages of an HC are apparent. It affords the ability to adapt to wide range of applications without redesign and has inherently high fault tolerance, due to the redundant multiplicity of machines, and increased longevity, due to its inherent flexibility.

Although several HC sites already exist, a full-scale HC environment is unlikely to be a feasible initial starting point for HPP application. Its obvious drawback is the high entry

costs — not just one expensive supercomputer must be purchased but several. In addition, high speed interconnection networks are still in development. Moreover, much research is needed to make possible interoperability of code and to provide tools for cross-compiling, execution profiling, and so forth. Although, eventually, partitioning and mapping may be handled by an intelligent component of the system, user direction is essential in the current state-of-the-art.

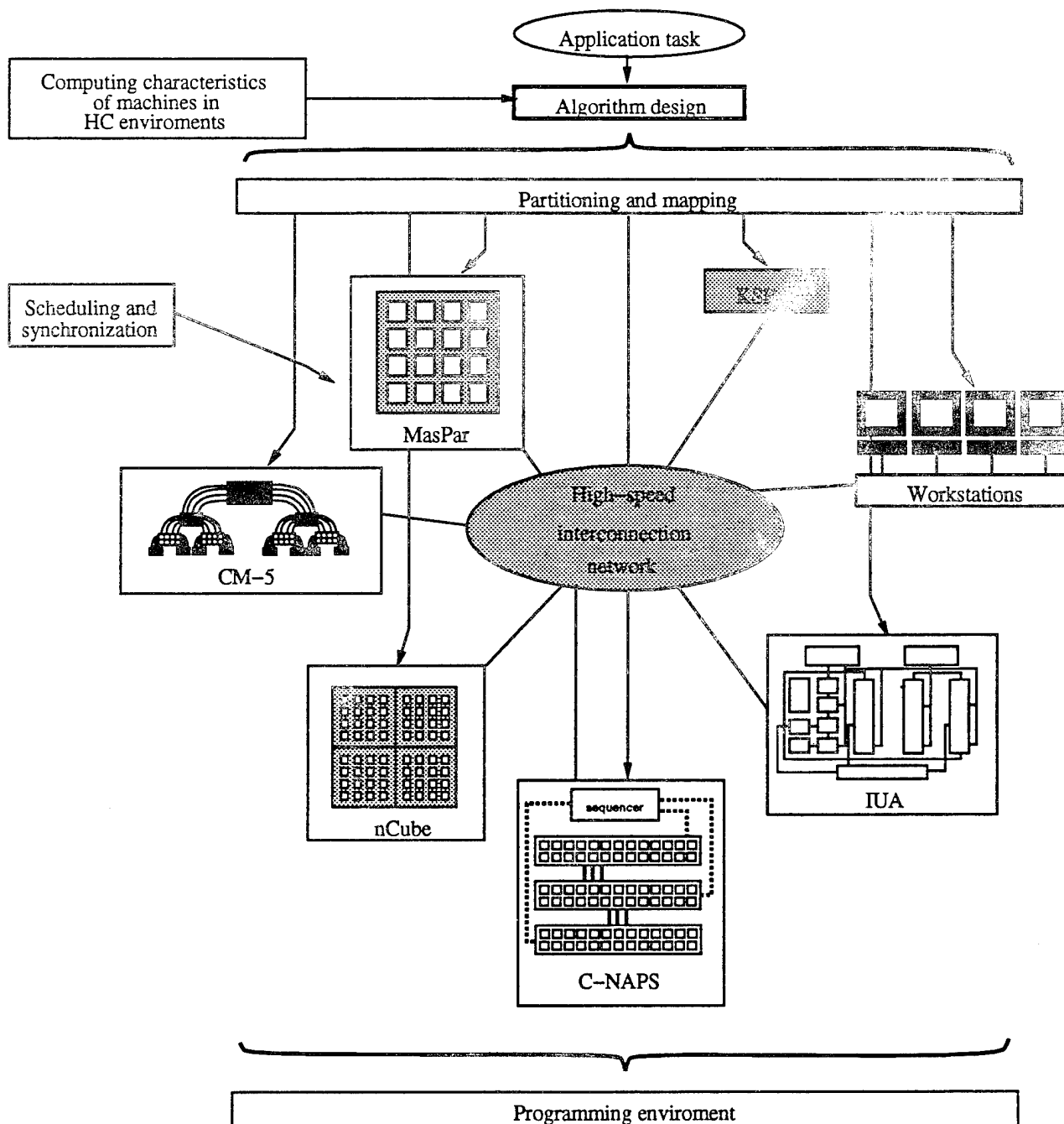


Figure 8.1: Heterogeneous Computing Environment

## Chapter 9

# Evaluation of Platforms

Evaluation of the platform alternatives presented in the previous chapter is summarized in Figures 9.1 through 9.6.

### 9.1 Scalability Performance Metrics

The information in Figures 9.5 and 9.6 figures was collected from Dongarra's "Performance of Various Computers Using Standard Linear Equations Software" [Dongarra-93]. The report presents the performance of different computers in solving dense systems of linear equations called Linpack Benchmarks. Linpack programs can be characterized as having a high percentage of floating-point arithmetic operations.

For each type of machine, the data indicates the theoretical peak performance in Mega Flops and the observed peak performance in Mega Flops for the largest problem run on the machine. The problem size is allowed to increase until it exceeds the machine's capacity. Therefore, the performance numbers demonstrate each machine's potential peak performance.<sup>1</sup> Although Dongarra's study is the best indicator of supercomputer ranking, such performance figures do not necessarily reflect overall system performance. One system may seem superior on some benchmarks while another may do better on other tests.<sup>2</sup>

*Conclusion:* Although non-linear saturation was a problem for earlier systems, the problem seems to have been overcome for the 0 — 100 Giga Flops range. Scalability is achieved by adding high bandwidth communication modules as systems grow larger.

---

<sup>1</sup>Theoretical peak performance is useful to indicate the maximum performance possible — it is not by itself an indicator of the actual performance achievable.

<sup>2</sup>NAS benchmark results are still not available on most of the machines considered here. Linpack measurements are the most commonly available numbers and are roughly representative of the results of other benchmarks on other machines. Therefore, the following conclusions are not invalidated by the use of Linpack measurements.

MACHINES	EVALUATION METRICS			
	Processing Elements (PEs)	Performance (MFLOPS)	Cost (\$M)	MFLOPS / \$M
CM-5	1024	59700	20	2985
iPSC/DELTA	512	13900	5	2780
GigaCube 3/1024	1024	16000	8	2000
MP-2	16K	1600	1	1600
MEIKO-CS-1	42	970	0.6	1600
KSR-1	256	3380	5	676
nCUBE-2S	1024	1900	3	633
Cray C90	16	13700	30.5	449
PARAMID *	8	1600	0.5	3200
PARAM *	256	1000	0.5	2000
LINDA W/SPARC	10	224	0.3 <sup>†</sup>	747
HP-9000	10	400	0.51 - 1.12 <sup>‡</sup>	356 - 734
DEC-3000	10	301	0.42 - 0.60 <sup>‡</sup>	498 - 716

<sup>†</sup> Cost includes \$3000 per node for LINDA

<sup>•</sup> This table contains summary data for several machines that were not included in figures 9.1, 9.2 and 9.6 for lack of sufficient data.

<sup>\*</sup> Performance data is not measured by LINPACK benchmark, but by the vendors' test results.

Figure 9.1: High Performance Systems Metrics

## 9.2 High Performance System Metrics

For the MPP platforms in Figure 9.1, the performance data indicate the observed performance in Mega Flops for the largest problem size taken from Dongarra's "Top 500 Supercomputers" [Dongarra-93a]. The cost data were quoted by the vendors or obtained from the managers at installation sites. The Mega Flops/\$M data is a ratio of observed peak performance to cost in units of millions of dollars. The asterisked (\*) computers are too recently developed to have been Linpack benchmarked. Vendor data make them very attractive platforms to be considered in the future. For example, the new PARAM supercomputer being developed with the T9000 Transputers is expected to achieve Tera Flop performance for approximately \$ 1 Million. The last three rows in the figure compare the different types of networked workstations combined with Linda language-level extender. Since the memory size and other factors determine the cost of workstations, the Mega Flops/\$M data are given as ranges.

*Conclusion:* The CM-5 has the best performance/cost ratio closely followed by the iPSC Delta.

MACHINES		EVALUATION METRICS					
		CPU	OS	Memory	Performance (MFLOPS)	Cost(US \$)	MFLOPS / \$M
Low-range	HP-9000 (715/33)	PA-7100 Chip	HP-UX	RAM:16MByte 128K Cache	8.6	6 - 40K	215 - 1433
	SPARCclassic	MicroSparc390 S10	Solaris 2.1	RAM: 16MB 64KB Cache	4.6	4 - 4.3K	1070 - 1150
	SGI R3000	MIPS R3000 with R30104 FPU	Unix SVR4	RAM:16MB 64KB Cache	4.3	7K	614
Mid-range	DEC-3000 (400 AXP)	AXP 21064, 133 MHz	OSF/1 or Open VMS	RAM:16MB 512KB Cache	26.4	15 - 37K	714 - 1760
	HP-9000 (715/50)	PA-7100	HP-UX 7.01	RAM:16MB 128KB Cache	13	12 - 55K	236 - 1083
	MirageIPS/10	Sparc 10	Solaris 2.1	RAM:16MB 64KB Cache	20	13 - 15K	1333 - 1538
	SGI Indigo 2 Extreme	MIPS R4000	Unix SVR4	RAM:32MB 16KB Cache	16	35K	457
	SUN SPARC10 41 GS	TI Supersparc	Solaris 2.1	RAM:32MB 36KB Cache	22.4	27K	830
High-end	DEC-3000 500 AXP	AXP 21064, 150 MHz	OSF/1 or Open VMS	RAM:32MB 1MB Cache	30.1	39 - 60K	502 - 772
	HP-9000 735 CRX-2YZ	PA-7100	HP-UX 9.01	RAM:32MB 256KB Cache	40	48 - 112K	357 - 833
	Intel Pro 6880	Clipper C400	Unix SVR 3	RAM:32MB 16KB Cache	16.3	34 - 57K	286 - 480
	SGI Onyx/Reality Engine 2	Two MIPS R4400	IRIX	RAM:64MB 32KB(int) and 1MB(ext) Cache	50	16.3K	3068

Figure 9.2: Workstation Price & Performance Metrics

### 9.3 Workstation Price & Performance Metrics

Figure 9.2 shows a representative sample of available workstations in three price categories. The Low range workstations are usually under \$10,000, the Mid-range falls between \$10,000-\$45,000, and the High-end computers are more than \$45,000. The figure shows the Linpack benchmarks compare favorably to those shown earlier for supercomputers [Furtney-93].

*Conclusion:* The DEC-3000 (400 APX) has the best performance/cost ratio among the high-performance systems in Figure 9.1. However, the Sun Sparc-10 has the advantage in software availability.

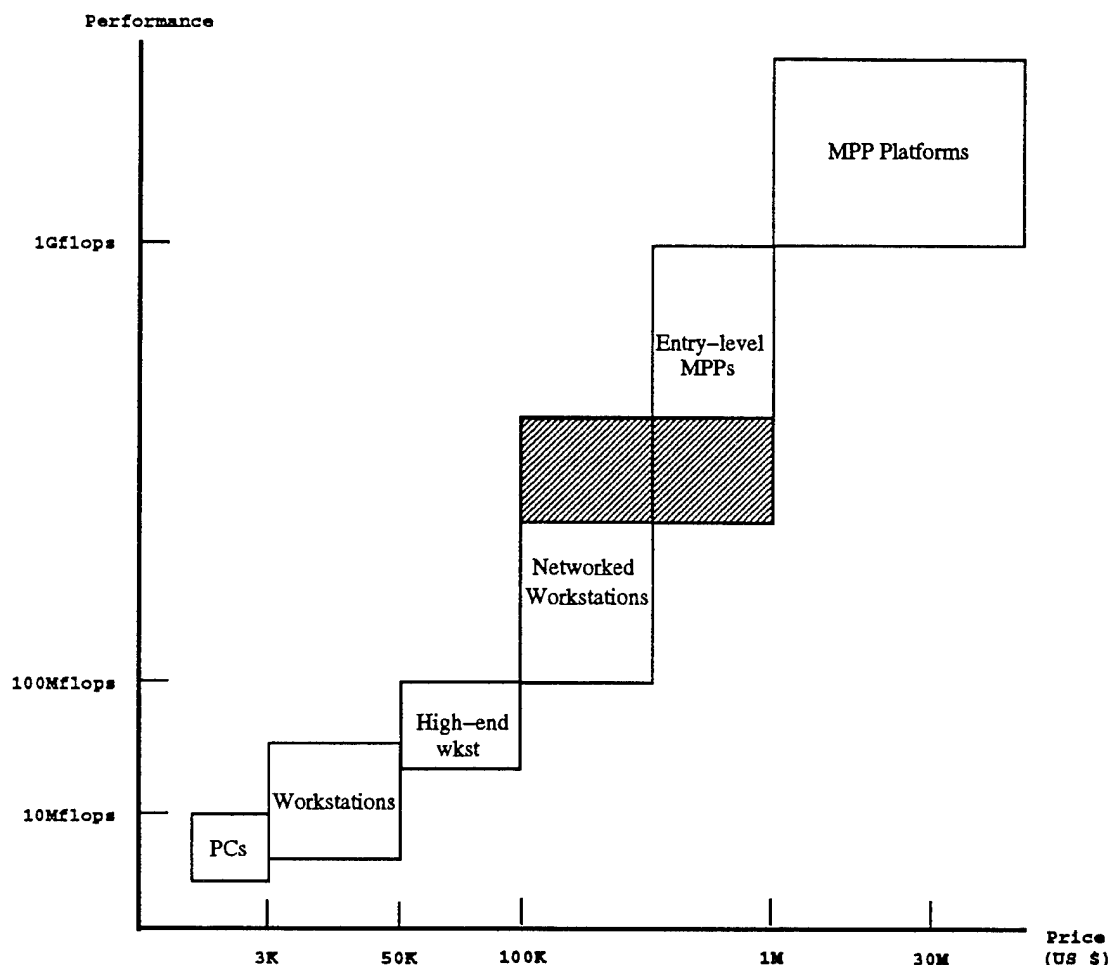


Figure 9.3: Price/Performance Baselines for Computer Hardware

## 9.4 Price/Performance Baselines for Computer Hardware

Figure 9.3 shows the borderlines of various computing resources in terms of their price and performance<sup>3</sup>. The data for this table were collected from the *comp.parallel* Newsgroup and Furtney's article [Furtney-93].

*Conclusion:* As expected, performance and price generally increase from personal computers (PC) to massively parallel processing systems. *However, networked workstations can achieve superior performance to entry-level Massively Parallel Processors (MPP) at a lower price.*

<sup>3</sup>Lower (upper) bounds were obtained by estimating the maximum number of nodes obtainable at the lower (upper) price level in combination with the worst (best) performance per node.

MACHINES	EVALUATION METRICS			
	Proc. Elem. (PEs)	Performance (MFLOPS)	Cost (\$M)	MFLOPS / \$M
iPSC/DELTA	8	230	0.2	1150
CM-5	32	1900	2	950
MP-2	1024	92	0.11	836
KSR-1	8	146	0.25	584
nCUBE-2S	64	121	0.33	367
PARAMID *	1	60	0.035	1714

\* Performance data is not measured by LINPACK benchmark, but by the vendors' test results.

Figure 9.4: Entry-Level Systems Metrics

## 9.5 Entry-level System Metrics

As shown in Figure 9.4, the number of processing elements for each system is the minimum configuration available in the real market. Entry-level configurations are based on data provided by vendors and from posting on the *comp.parallel* Newsgroup.

*Conclusion:* The iPSC-Delta has the best performance/cost ratio and the lowest cost of the benchmarked entry level systems.

## 9.6 Recommendations

No one single computer (within the foreseeable choices) is likely to meet the requirements of HPP model development because the differing functions within the cognitive, sensory, memory, and motor components are best emulated on parallel architectures with distinct styles of processing. Moreover, purchasing a high performance parallel platform would entail a large capital outlay that would be difficult to justify, given the goal of transitioning from a system such as OMAR developed on a single conventional workstation. Instead, an evolutionary approach should be adopted that can transition from an initial workstation to a network of workstations (or even to a heterogeneous network of computing nodes) as the modeling effort evolves in its demands for increasing resources.

Summarizing the previous conclusions, the following network of workstations is recommended<sup>4</sup>: starting with two Linda connected mid-range (DEC 400 APX) workstations [Furtney-93] and expanding to a maximum of 16 (tested thus far) for the initial (first level) network [Kaashoek-89, Ahuja-86, Leler-90]. Subsequent hierarchical expandability enables an unlimited number of workstations (heterogeneous computing environments) to be connected with a maximum of 16 at each level.

*Initial Cost* (2 workstations & Linda): approximately \$36K.

*Cost* (10 workstations & Linda): approximately \$ 180-400K.

*Peak Linpack Performance*: 264 MFlops.

In the next chapter, an approach to evolving HPP models that can be supported by networks of workstations is desired.

---

<sup>4</sup>These recommendations are subject to revision as the state-of-the-art in heterogeneous computer environments matures. When the hardware and software support for HCE becomes fully usable, it will be appropriate to reanalyze the questions addressed here in terms of both cost and optimality-for-task.

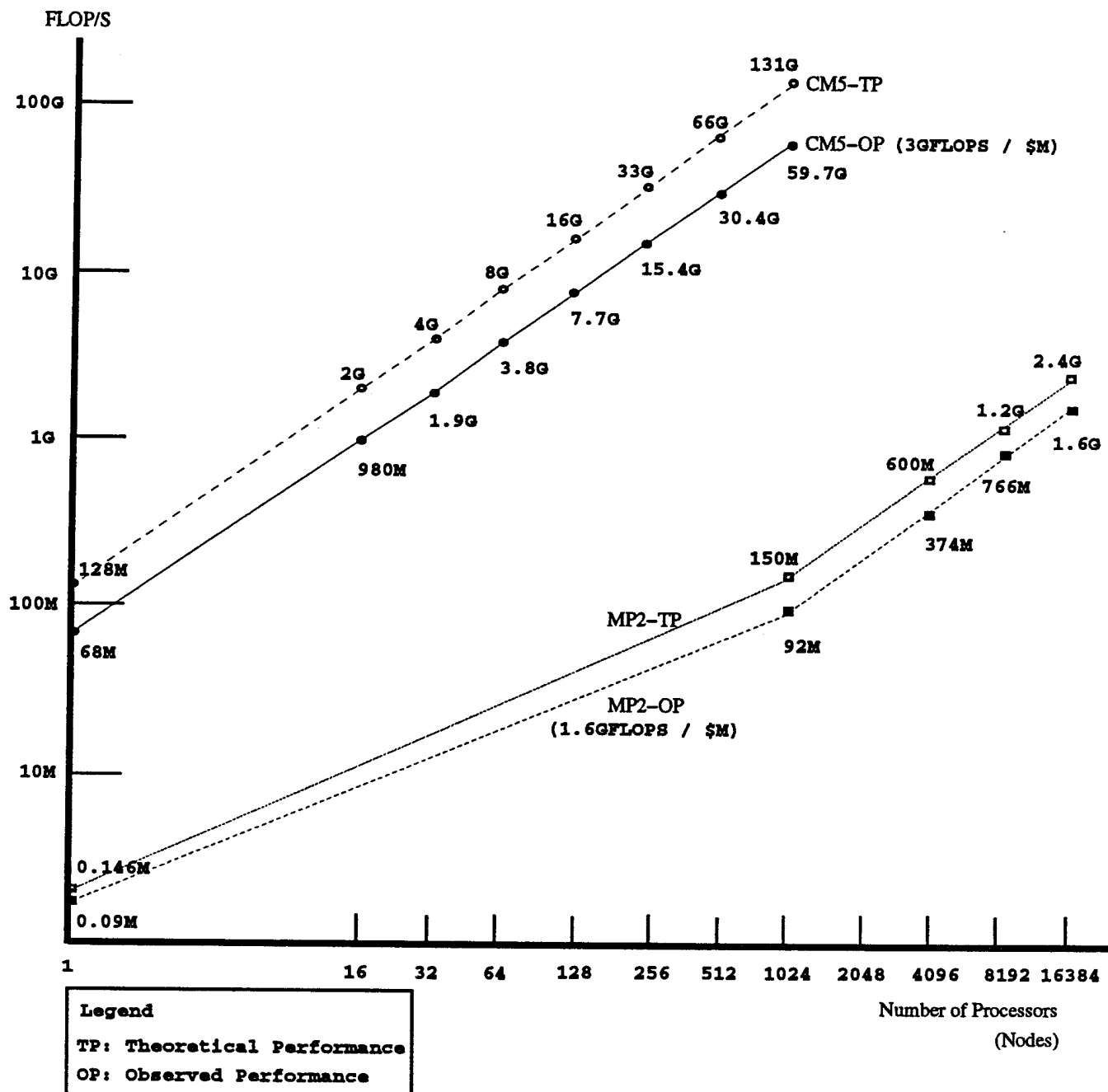


Figure 9.5: Scalability Performance Metrics of Data Parallel Platforms

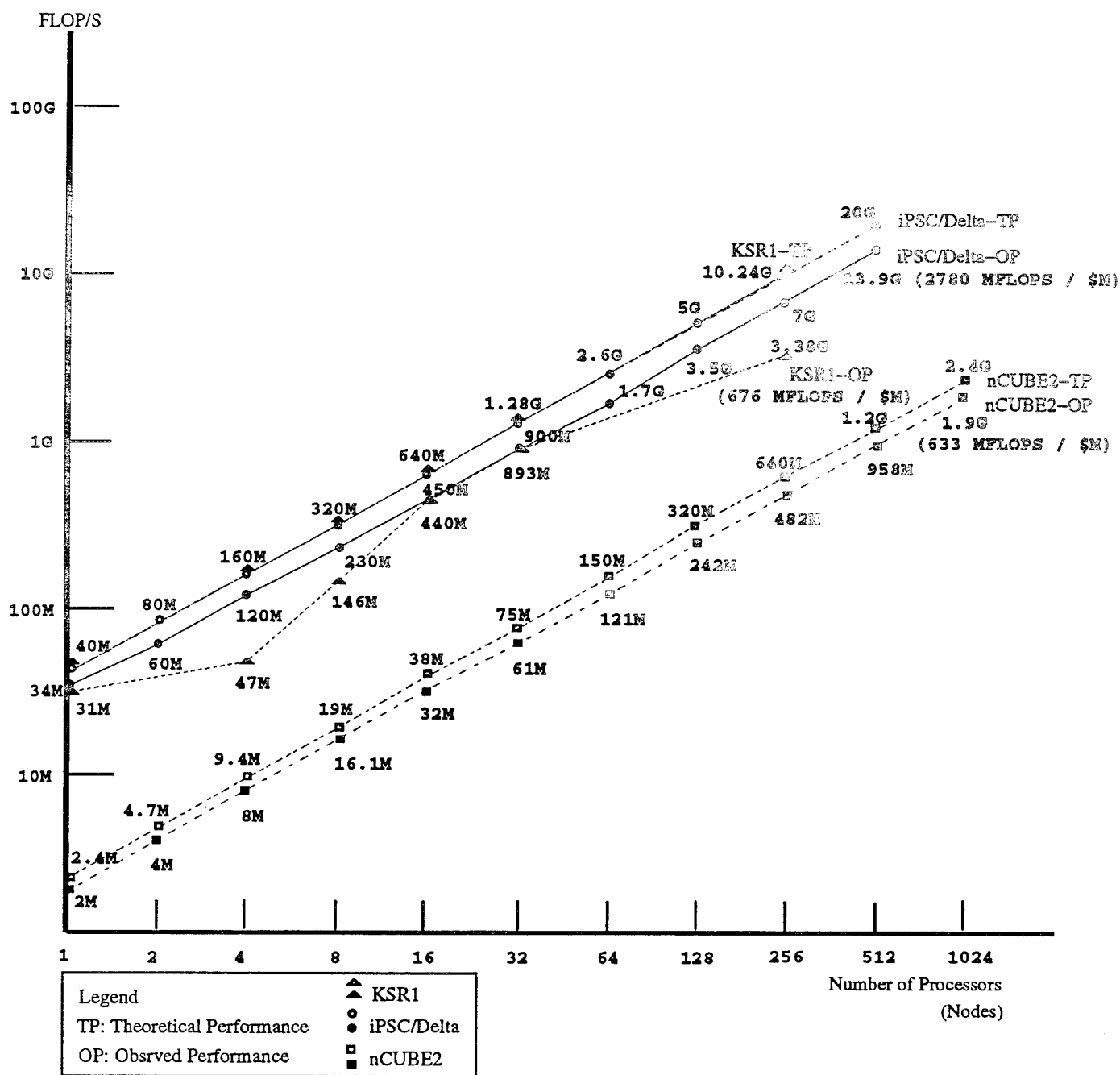


Figure 9.6: Scalability Performance Metrics of Multiple Instruction Multiple Data (MIMD) Platforms

## Chapter 10

# Hierarchical Evolution of Human Performance Process Models & Platforms: Successive Approximation

As stated earlier, no one single computer (in the near future) is likely to meet the requirements of HPP model development because the differing functions within the cognitive, sensory, memory, and motor components are best emulated on parallel architectures with distinct styles of processing. Moreover, purchasing a high performance parallel platform would entail a large capital outlay that would be difficult to justify given the goal of transitioning from a system such as OMAR developed on a single conventional workstation. Instead, an evolutionary approach that can transition from an initial workstation to a network of workstations (or even to a heterogeneous network of computing nodes) as the modeling effort evolves in its demands for increasing resources should be adopted.

An approach to model development based on hierarchical decomposition that would guide coherent model evolution and, at the same time, enable the simulation platform to co-evolve in a matching, "continuous" manner is also suggested. The ability of a computing system to grow in such a manner is called *hierarchical expandability*. It also happens that hierarchical expandability of platforms meshes well with the method of successive approximation advocated by [Young-92] for the development of HPP models. In this approach, the incremental development methodology of systems engineering is followed such that during each development cycle, a complete HPP model is developed, tested, and validated. The method begins with the development of an initial domain model that is fairly comprehensive in scope, albeit limited in the range of behaviors it can accurately reproduce. Moreover, many processes in initial models may be depicted at low levels of resolution or may be absent altogether. Through successive increments of development, the behavioral range of the model is extended by increasing its resolution.

Successive approximation concepts can be formalized by employing the systems entity structure to represent model evolution [Zeigler-95]. Visualizing the model development process

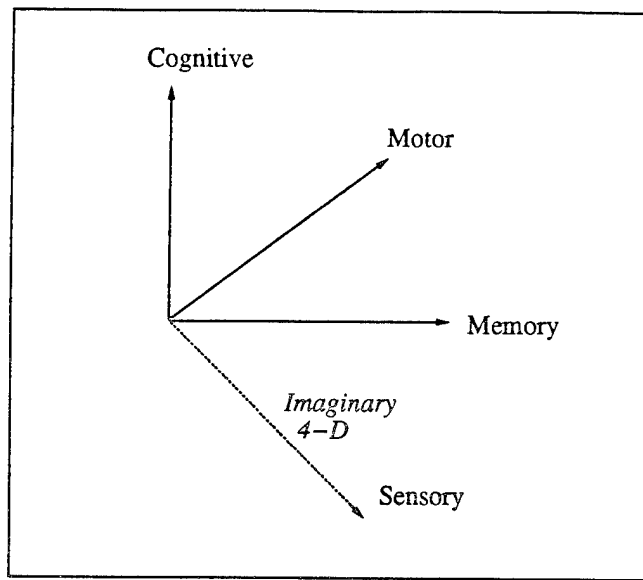


Figure 10.1: Human Performance Process Model Space

as a trajectory in model space (see Figure 10.1) will help explain the approach.

Figure 10.1 illustrates the main dimensions in HPP modeling space. Any HPP model can be thought of as being represented as a point in this space, having certain mechanisms for each of the dimensions: memory, cognition, motor, and sensory. (Indeed, these mechanisms may derive from the metaphor approach to synthesis discussed earlier.) Each axis points in the direction of increasing complexity so that points close to the origin represent models of relatively low complexity, and points further out represent relatively high complexity. The evolution of HPP modeling can be imagined as path in this space, starting near the origin and generally progressing outward, though not necessarily in a straight line fashion. Two forms of movement in this space can be discerned:

- *Exploration of Model Alternatives:* In this case, different versions of the same mechanisms are tested, generally without much change in complexity levels.
- *Alterations in Model Resolution:* Most often, elaboration or refinement in representation tends to increase complexity, but sometimes the inverse process of abstraction tends to reduce complexity.

Such progression is discussed in greater detail in Chapter 11.

Figure 10.2 illustrates how the implementations of HPP models on computer platforms can be conceived of as a mapping from HPP model space to a space representing computer architectures. This mapping should have a continuity to it in the following sense: when an initial model is implemented on an initial platform (e.g. OMAR implemented on a Sparc workstation) and the initial model evolves through model space, the initially chosen architecture should be adaptable to accommodate it without requiring a radical change to

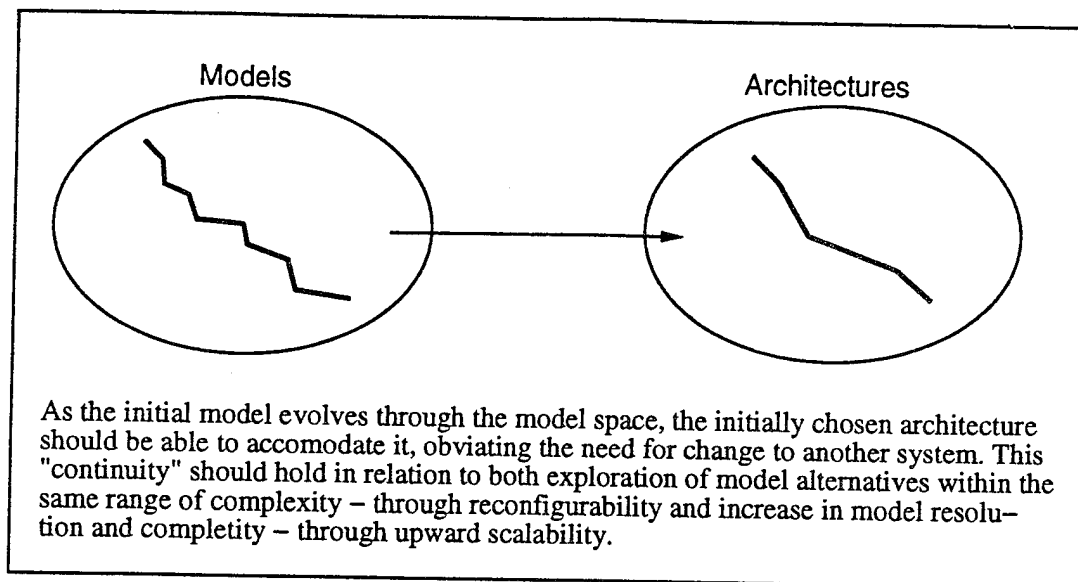
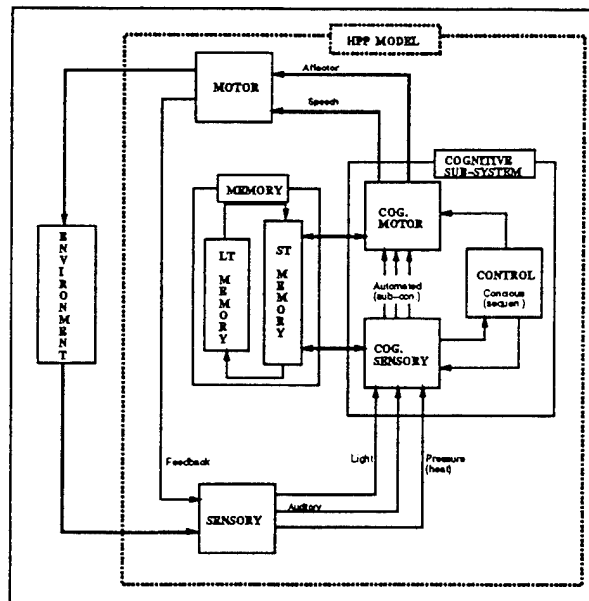


Figure 10.2: "Continuous" Mapping: Models to Architectures.

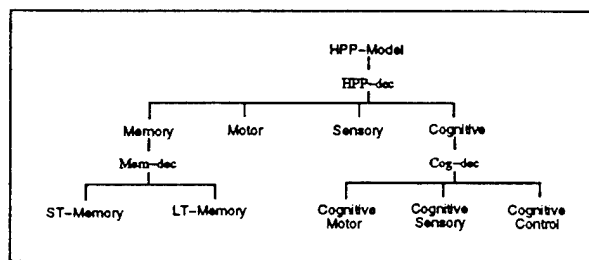
another platform. This "continuity" should hold in relation to both exploration of model alternatives within the same range of complexity and increase in model resolution tending to increase complexity. In the case of exploration of alternatives, this continuity may require that the platform be reconfigurable to maintain the desired level of simulation performance. In the case of elaboration of the model structure, it may require that the platform be scalable (i.e., accommodate the extra resources required to handle the greater complexity). Thus, in a "continuous" relationship between model and platform, a path of model development supported by a continuously evolving platform without too radical a change in its basic architecture would be seen over several years.

Model development can be conceived in terms of hierarchical decomposition and described using System Entity Structure (SES) concepts (briefly reviewed in Appendix D.3.4). Figure 10.3 shows a block diagram (a) of prototype HPP model and a corresponding SES (b) that represents it (actually, the coupling relationships describing the interconnections between model components are not shown in the SES). Model refinement can be thought of as represented by a top-down development of the SES, in which blocks of the basic prototype are successively broken open into subcomponents. An example sequence of such hierarchical decomposition is illustrated in successive SESs given in Appendix E. Of course, while such hierarchical decomposition may be the overall trend, as already indicated, model development need not always move toward greater complexity.

Figure 10.4 illustrates some operations on SESs having various effects on complexity. In the grouping (also called flattening) operation (Figure 10.4a), some of the components of a model are merged into a larger, equivalent, component. This operation is the inverse of decomposition (Figure 10.4b), in which a component is replaced by a composition of lower level components. Exploration of alternatives can be viewed as the operation of providing variants for a component. The bold arrow in Figure 10.4c indicates that either of the



(a) Block Diagram



(b) System Entity Structure (SES)

Figure 10.3: Human Performance Processing Model Prototype

variants B1 or B2 may be substituted in the slot occupied by component B in model M. Other operations, such as distributing the functionality of a component among its peers, can also be represented as in Figure 10.4d.

As mentioned, model evolution conceived as a sequence of operations on an initial SES is illustrated in Appendix E. Starting with the initial SES in Figure 10.4, the first operation is to distribute memory among the cognitive, sensory, and motor components, meaning that memory does not stand alone as a distinct storage component, but is intimately connected to each of the functions which it supports. Next, the cognitive system is decomposed into parts dealing with memory, sensory processing, motor output, and cognitive control. The original cognitive system is shown as still employable as an alternative. The reader may wish to continue examining the progression. Of course, it is meant to be illustrative, not definitive, of how evolution might occur in HPP model development.

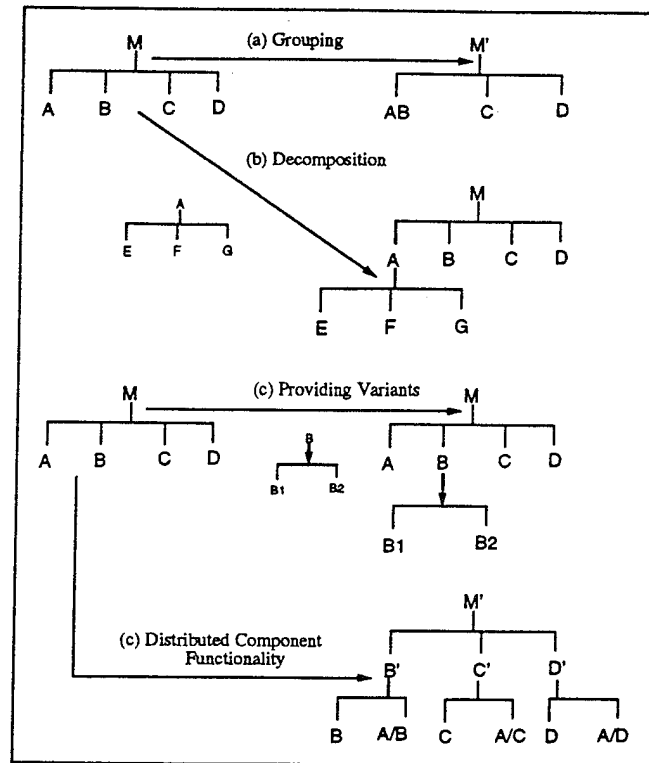


Figure 10.4: Operations on the System Entity Structure

Given this sketch of HPP model evolution, an architecture is *hierarchically expandable* if it supports such evolution in the "continuous" fashion discussed earlier. This requirement is more specific and demanding than the conventional concept of scalability for parallel platforms; where typically, a platform is viewed as growing linearly with addition of more processors. However, in this context, it is most natural to require that the platform expand in a fashion retaining structural correspondence with successive model decompositions. Moreover, such hierarchical expandability should offer an effective and flexible approach to meeting heterogeneity requirements, especially when coupled with the demands for real-time model operation imposed by man-in-the-loop simulations.

A network of computing nodes with support for attachment of subnetworks, offers a plausible architecture to implement hierarchical expandability. As illustrated in Figure 10.5, each node in a network handles a top level model component. To support a new decomposition, the node that handled the original component is connected to a subnetwork connecting its new components; of course, the node also retains the original interface to its peers. Whereas originally the node itself handled the response of its model component to inputs, it now transmits these inputs to the attached cluster and relays its response back to the other components at its level. The clustered network in Figure 10.5 is the result of employing decompositions at Level 2 for the sensory and motor components and at Level 3 for the cognitive component.

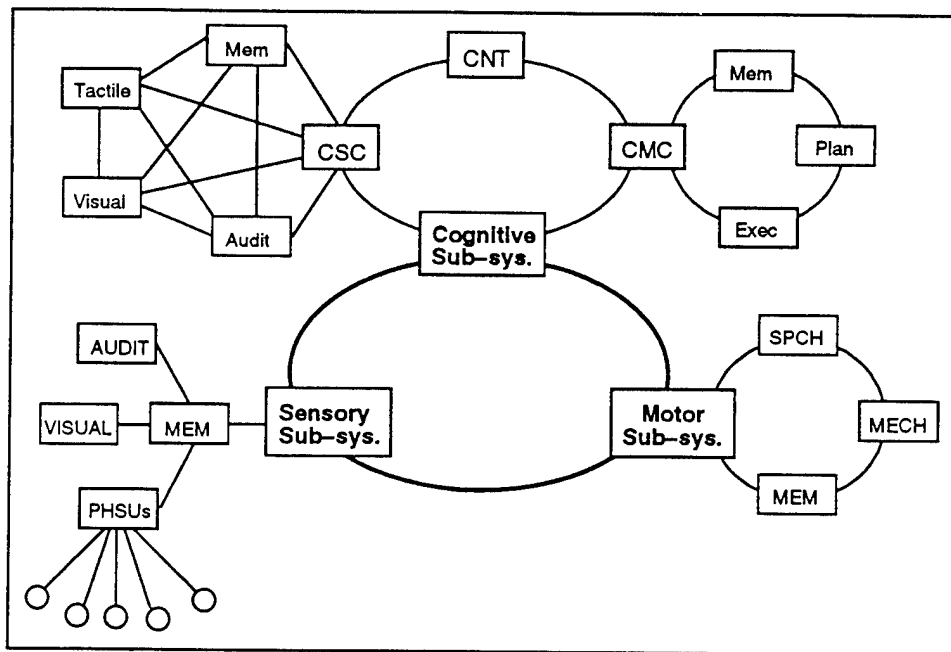
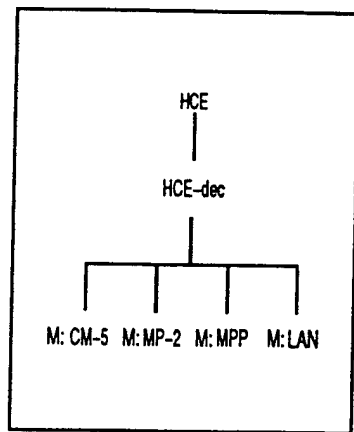


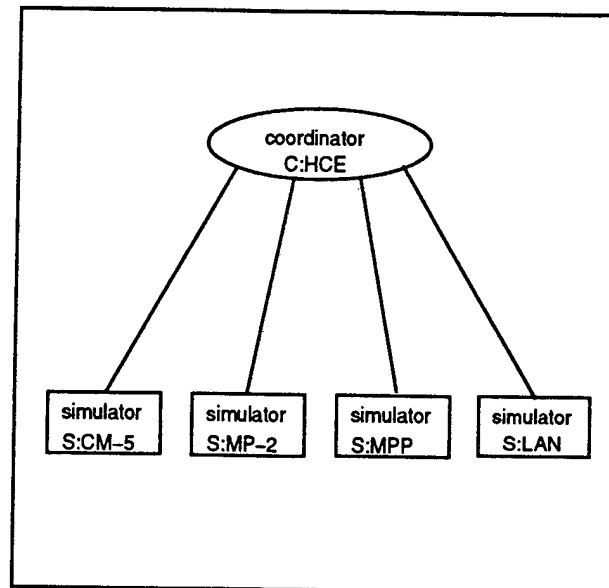
Figure 10.5: Evolutionary Network of Computing Nodes

The DEVS abstract simulator (Appendix D.3.5) provides a ready-made logic for realizing hierarchical expandability. Figure 10.6a illustrates how an initial model decomposition is first mapped into a corresponding abstract simulator that simulates it. This abstract structure is realized by a network of workstations including those for the simulators of model components and one for the coordinator that ensures the validity of overall simulation. For concreteness, it is assumed that Scalable Coherent Interfaces (SCI) connect the nodes to a high speed ring network. Figure 10.6b illustrates how a decomposition of a model component is reflected in corresponding extension of the abstract simulator and expansion of the network implementation. Note that a bridge is employed to link a new subnetwork into the proper slot of the original network (Figure 10.7). An intriguing possibility would be to leave the original simulator in the network with a switch that accepts the faster of the two responses from the original model component or its clustered decomposition, amounting to a representation of automated and deliberate response competition.

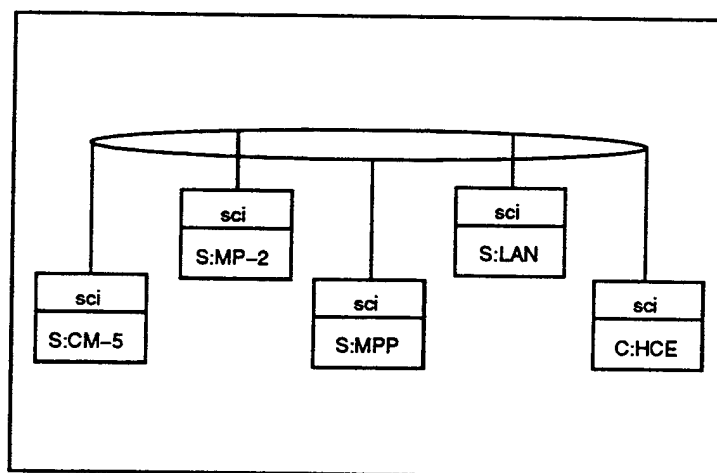
This approach has natural **load-balancing benefits**: A level of decomposition can be undertaken not only to achieve increased model resolution, but also to improve performance for real-time or non real-time application. For increased resolution, the newly introduced components require additional resources if performance is to be maintained. Decomposition to achieve performance enhancement may be done to exploit parallelism within the model component. It requires replacing, or augmenting, a single computer with several processors. In either case, the hierarchical expansion concentrates resources where they are needed, namely within a cluster dedicated to the new decomposition.



(a) SES Representation

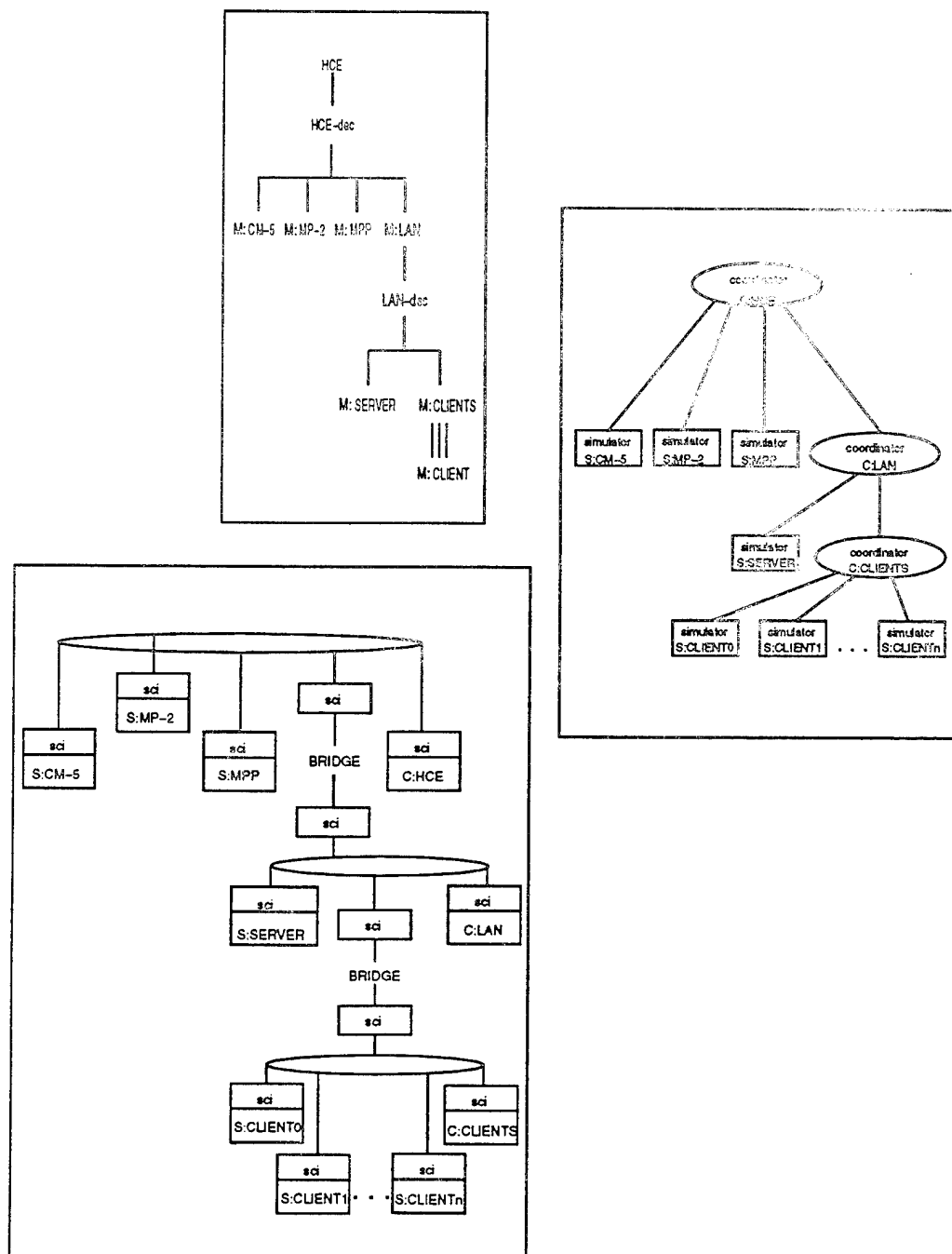


(b) Abstract Simulator



(c) SCI Ring Network

Figure 10.6: Hierarchical Evolution I



## Chapter 11

# Follow-on Investigation

In conclusion, no one computer (within the foreseeable choices) is likely to meet the requirements of HPP model development. Instead, an evolutionary approach that can transition from an initial workstation to a network of workstations (or even to a heterogeneous network of computing nodes) as the modeling effort evolves in its demands for increasing resources should be adopted. An approach to model development based on hierarchical decomposition that would guide coherent model evolution and at the same time enable the simulation platform to co-evolve in a "continuous" manner has been recommended. The ability of a computing system to grow in a such a manner is called hierarchical expandability and is identified as a more specific and demanding requirement than the conventional concept of scalability for parallel platforms. Hierarchical expandability would offer an effective and flexible approach to meeting heterogeneity requirements, especially when coupled with the demands for real-time model operation imposed by man-in-the-loop simulations.

While the initial feasibility of the hierarchical evolution approach was established, much work is still required before it can be put into practice confidently. The first step would be a detailed study to propose alternative designs and investigate their feasibility. Since hierarchical distributed simulation is well-understood from prior work, the main issues to be addressed here involve the specific demands of hierarchical expandability, particularly in the context of real-time distributed simulation, and the design of supporting platforms from off-the-shelf software and hardware. In other words, we need to know more about how to construct distributed simulations to operate in real-time, how to expand such simulations hierarchically while retaining real time performance, and how to implement these capabilities with commercially available, or foreseeable, networking primitives. The study should address such issues as the following.

- Algorithms for real-time hierarchical distributed simulation.
- Procedures for expanding a node into a subordinate cluster.
- Criteria for node expansion to meet imposed time constraints.
- Potential software primitives (distributed operating systems, messaging systems, and distributed programming systems).

- Potential hardware primitives (high performance network interconnects).
- Implementation approaches for algorithms and expansion procedures employing hardware and software primitives.
- Approach to non-real-time requirements by relaxing real-time criteria.

Extension of hierarchical evolution application to facilitate holon-based HPP models will require further decomposition into hierarchical (heteroarchival), fine-grained holarchies. Related issues are as follow.

- Network layering.
- Communication patterns within and between layers.
- Dualistic polymorphism of holons.
- Relation to HPP-oriented neural networks.

## Appendix A

# Human Performance Process Metaphors

Engineering Psychology [Wickens-84]:

- **Intelligence/Competence aspired to:** human operator (manual control, tracking, decision making, signal detection, navigation, fault diagnosis).
- **Performance attributes under consideration:**
  - Dimensions: efficiency, bias, optimality, reaction time, latency, accuracy, speed accuracy tradeoff, information transmission rate.
  - Errors: misperception, mode, slips, performance operating characteristic (POC), performance-resource function (PRF) practice, task difficulty, receiver operating characteristic (ROC).
  - Effects of: mental workload difficulty, composition, stress, individual differences.
- **Integration with sensing/action in the real world.**
- **Built-in structure:** information processing model.
  - *Primitives:*
    - Processing elements.
    - Stages: encoding, central processing, responding.
    - Modalities: visual, auditory.
    - Codes: auditory (echoic, phonetic), visual (iconic, visual primary), semantic, spatial, verbal.
    - Responses: manual, vocal
    - Resources.
  - *Architectures:*
    - Multiple resource theory: resource allocation to processes.
    - Supported by memory: short-term sensory store, working, long-term.
    - Capacity constraints on: processing and memory.
- **Emergent Behavior:**
  - Laws: Fitts, Hick-Hyman.

- Automatization levels: skill-based, rule based, knowledge based, attention(selective, focused, divided), time-sharing, psychological refractory period (PRP)
- Processing: serial, parallel.
- **Learning Mechanisms.**
- **Coordination Mechanisms.**
- **Dynamic/Temporal Considerations.:** transfer functions, time domain, frequency domain, crossover model, optimal control model, serial RT paradigm.

Neural Darwinism [Edelman-87, Damasio-89]:

- **Intelligence/Competence aspired to:** survival in real world.
- **Performance attributes under consideration:** diseases of consciousness.
- **Integration with sensing/action in the real world:** continuous, tightly coupled integration.
- **Built-in structure:**
  - *Primitives:*
    - Neuronal groups (each representing a population of 50-10,000 neurons with gene pool-like variability).
    - Reentrant signaling (representing intrinsic (within group) and extrinsic connectivity).
    - Hedonistic value system.
  - *Architectures:*
    - Low-level: neuronal group properties and reentrant connectivity postulated to conform to anatomically identified neuroanatomical structures and pathways (convergent, divergent, registered, arborized, layered).
    - High-level: maps (sensor receptor sheets to nervous system regions).
- **Emergent Behavior:**
  - Low-level: generalization, categorization results from correlation of neuronal group firings.
  - High-level: memory as recategorization, primary and secondary consciousness, attention.
- **Learning Mechanisms:** neuronal group selection (pre- and post-synaptic amplification of synaptic populations).
- **Coordination Mechanisms:** Resolution of conflicts: reentrant circuit acts as censor.
  - Cross-modal construction: one area uses outputs of another for its own functions.
  - Recursive synthesis: outputs of higher area influence inputs of lower area.
- **Dynamic/Temporal Considerations:**
  - Group stability.
  - Signaling: phasic, cyclic, synchronous, asynchronous.
  - Lower level response superseded by higher level if not generated fast enough.

### **Society of Mind [Minsky-86]:**

- **Intelligence/Competence aspired to:** higher-order thinking, problem-solving.
- **Performance attributes under consideration:** none apparent.
- **Integration with sensing/action in the real world:** partial.
- **Built-in structure:**
  - *Primitives:*  
Agents (behavior simple enough to understand - might be group of neurons)  
Specialized connector-agents: K-lines, Nemes, Nomes  
Frame-arrays.
  - *Architectures:*  
Low-level: agencies (hierarchical compositions of agents).  
High-level: heterogeneous clumping of agencies with sparse interconnections.
- **Emergent Behavior:**  
Low-level: recognizers, memorizers, difference-engines.  
High-level: visual ambiguity, language understanding, serial processing made possible by recursive memory as better than distributed processing.
- **Learning Mechanisms:** accumulating, unframing, reformulating, trans-framing, modeling.
- **Coordination Mechanisms:**  
Resolution of conflicts: principle of noncompromise.  
Interruptability, recursive (stack-like) memory.  
Suppressor-agents, censor-agents.  
Exploitation of agents using model.
- **Dynamic/Temporal Considerations:** time blinking.

### **Fallible Machine [Reason-90, Anderson-90, Newell-90]:**

- **Intelligence/Competence aspired to:** common knowledge question answering (cognitive symbol level behavior).
- **Performance attributes under consideration:** errors in output.
- **Integration with sensing/action in the real world:** none.
- **Built-in structure:**
  - *Primitives:*  
Peripheral Working Memory (PWM).  
Focal Working Memory (FWM) (production rule engine).  
Buffer store.  
Knowledge base (KB), procedural and declarative elements and composites (units, spreading activation, hierarchical problem space decomposition capability).
  - *Architectures:*  
Low-level: sensory input to PWM and KB.  
PWM buffers FWM by prioritized input intrusion.

FWM serial inferencer, outputs to buffer store.

Buffer store queries KB.

KB parallel retrieval based on similarity matching and activation level (frequency gambling).

- **Emergent Behavior:**

Low-level: Error prone question-answering due to frequency gambling.

Wandering attention due to resources needed to maintain current buffer against active KB units.

- **Learning Mechanisms:** FWM compiles co-occurring knowledge elements into units.

- **Coordination Mechanisms:** prioritization, knowledge unit compilation, buffer intrusion suppression.

- **Dynamic/Temporal Considerations:** time sliced operation of FWM (3 elements/cycle of a few ms.).

#### Distributed Operating Systems [Sha-93, Tanenbaum-92]:

- **Intelligence/Competence aspired to:** guaranteed forward progress (non-real-time) or gracefully degrading (real-time) execution of user processes.

- **Performance attributes under consideration:** throughput, job completion time, utilization of resources.

- **Integration with sensing/action in the real world:** none.

- **Built-in structure:**

- *Primitives:* buffers, synchronization mechanisms, process state saving/restoration, scheduler.

- *Architectures:*

- Low-level: operating system kernel linked to network providing interprocess communication.

- **Emergent Behavior:**

Low-level: competence achievement.

- **Learning Mechanisms**

- **Coordination Mechanisms:** synchronization, prioritization, load balancing, flow control.

- **Dynamic/Temporal Considerations:** performance transient/steady state models.

#### Distributed AI [Lesser-90]:

- **Intelligence/Competence aspired to:** collective problem solving.

- **Performance attributes under consideration:** none.

- **Integration with sensing/action in the real world:** none.

- **Built-in structure:**
  - *Primitives:*
    - Agents (high level problem solvers).
    - Shared communication medium.
    - Blackboard (in some configurations).
  - *Architectures:*
    - Low-level: communications network.
- **Emergent Behavior:**
  - Low-level: cooperation to solve problems.
- **Learning Mechanisms.**
- **Coordination Mechanisms:**
  - Distributed, localized: information-sharing, negotiation, contracting, delegation of authority.
  - Abstraction constrained (top-down/bottom-up) processing.
- **Dynamic/Temporal Considerations.**

#### High Autonomy Systems [Antsaklis-89, Albus-90]:

- **Intelligence/Competence aspired to:** autonomous, adaptive, fault-tolerant plant operation.
- **Performance attributes under consideration:** meet cost, resource, time bounds.
- **Integration with sensing/action in the real world:**
  - Sensors: electromechanical, chemical, sonar, etc.
  - Sensor fusion and interpretation.
  - Effectors: actuators, switches.
- **Built-in structure:**
  - *Primitives:*
    - State-described processes.
    - Models of plant and own components.
    - Knowledge data-base.
    - Value assigner.
  - *Architectures:*
    - Low-level: planning, scheduling, execution, fault management hierarchies.
- **Emergent Behavior:**
  - Low-level: goal attainment, reactive recovery from anomalies and malfunctions.
- **Learning Mechanisms:** case-based reasoning.
- **Coordination Mechanisms:** hierarchical control.
- **Dynamic/Temporal Considerations:** time horizon associated with hierarchical level.

## Intelligent Robotics [Saridis-83]:

- **Intelligence/Competence aspired to:** visually navigate semi-structured environment and manipulate objects in it.
- **Performance attributes under consideration:** meet cost, resource, time bounds.
- **Integration with sensing/action in the real world:**
  - Sensors: electromechanical, chemical, sonar, and so forth.  
Sensor fusion and interpretation.
  - Effectors: servomotors, legs, wheels, arms, manipulators.
- **Built-in structure:**
  - *Primitives:*  
Goal search mechanisms.  
Maps/spatial representations.
  - *Architectures:*  
Low-level: trajectory/path planning, scheduling, execution.
- **Emergent Behavior:**  
Low-level: collision avoidance, movement/manipulation, goal attainment.
- **Learning Mechanisms:** neural networks (back propagation etc.), in vision and manipulation.
- **Coordination Mechanisms:** fully preplanned motion requires no coordination in predictable environment.
- **Dynamic/Temporal Considerations:** kinetic, mechanical models of effectors and objects in space.

## Appendix B

# Synopses of Selected Languages/Environments

**ART\*Enterprise (Inference Corp.):** An environment for the development of intelligent systems.

- **Functionality:**

1. Object-oriented Programming.
  - Object-oriented data modeling.
  - Multiple Inheritance with defaults and exceptions.
  - Single and Multiple Valued Attributes.
  - Polymorphism.
  - Generic Functions.
  - Methods.
  - Method combination.
  - Encapsulation.
  - Reusable Core Classes.
2. Procedural Language.
  - Data types.
  - Symbols, strings, numbers, sequences, objects, hash arrays.
  - Branching and looping constructs.
  - Recursive Functions.
  - Functions of variable number of arguments.
3. Rule-based programming.
  - Forward-Chaining.
  - Rete Algorithm Inference.
4. Hypothetical Reasoning with Consistency Management.
5. Case-Based Retrieval.
  - Built-in class definitions for case and case base.

- Methods for creating cases.
  - Methods for accessing and updating cases bases.
  - Case-based similarity matching.
- **Integration with Data Bases.**
- **Integration with Ansi C, C++.**
- **Development and Deployment Platforms:** UNIX, MVS, PC Operating Systems.
- **Price:** \$10,000.

**ProKAPPA (Intellicorp):** Language for intelligent system tool development.

- **Functionality:**
  1. **Object Manager:**
    - Classes, instances, slots (attributes), persistent/dynamic.
    - Single and multiple value slots and facets.
    - Operations for creating and modifying objects, slots, facets.
    - Multiple Inheritance with conflict resolution.
    - Demons.
    - Methods in C or ProTalk.
    - Function Libraries.
  2. **Pro-Talk Language:**
    - combining procedural, 4GL, OO.
    - backtracking engine.
    - pattern matching engine.
  3. **Rule-based Environment:**
    - Forward and Backward Chaining.
    - Rule agendas and priorities.
    - Rule Sets, Rule Classes.
    - Conditional Rules.
- **Active Relations.**
- **Integration with Data Bases.**
- **Integration with ANSI C, C++.**
- **Development and Deployment Platforms:** Sun/Sparc, HP, IBM RS, Sequent(in process).
- **Price:** \$20,000

**SimKit (Intellicorp):** Knowledge-based simulation in Knowledge Engineering Environment(KEE).

- **Functionality:**

- Lisp-Based simulation environment.
- Object-Oriented Programming.
- Fame-based Knowledge Representation.
- Demons, Active values.
- Rule-based Programming.
- Discrete Event Simulation: Clock, Calendar, Simulator, Generators, and Data Collectors.
- Composite Objects.
- Composite Objects.
- Relations between Model Instances.
- Model Library.
- Verification Slots.

**SCORE/SPROCKET:** Simulation Core Language (SCORE) is a discrete event simulation language developed at BBN [Abrett-70] and subsequently extended to SCORE/SPROCKET [Palmucci-92]. It features a Global Event Scheduling Queue, an efficient time management approach employed by most discrete event language implementations developed for conventional sequential processing. As one of the first languages to combine AI knowledge representation and conventional simulation concepts, SCORE, and more particularly SCORE/SPROCKET, contains several sublanguages that support planning and execution: Goal-Plan-Procedure Language (GPP), a Rule Language, and Very Simple Frame Language (VSFL). Further, it provides object-oriented procedure classes suitable for task prioritization in a multithreaded environment. Multiple agents execute procedures which can spawn subprocesses under sequential, parallel, race and join coordination schemes. Control of concurrent operation of processes can be exerted through primitives enabling suspension, resumption, and synchronization for waiting and interrupt handling. Constraints that are dynamically sensitive to procedure prioritization can be specified. Trace facilities are provided to examine event causality at various abstraction levels. SCORE/SPROCKET is written in CLOS and can be compiled for faster execution. It employs Common Lisp Interface Manager (CLIM) for graphic user interfacing. It runs on Macintosh, Sun, and Symbolics systems.

## Appendix C

# Neural Networks: An Overview

Recent years have seen an enormous upwelling of interest in the advancement and development of *artificial neural networks* [Caudill-87]. Before proceeding with the overview of neural networks, this section reviews some definitions of key terms.

*Neural Network*: an implementation of a learning algorithm based on neuroscientific research. These artificial networks typically contain one or more *layers* of artificial neurons composed of *weights*, *connections*, and *nodes*.

*Nodes*: single neuron-like elements that serve as the computing elements in the neural network.

*Layer*: a set of connected nodes, having common inputs, outputs, or both.

*Connections*: the links between the nodes used for communication and data transfer. Connections are typically weighted.

*Weights*: variables associated with the connections between nodes that are representative of the influence of the sending node on the receiving node.

Neural networks have demonstrated powerful pattern recognition abilities. The popularity of ANN stems from their natural ability to behave in ways that are otherwise computationally difficult to model. Some of these abilities are as follows:

- ANN can *detect the underlying organization and relationships* between the data provided without requiring any explanation of the data.
- ANN have the ability to *generalize* (i.e., respond to patterns that may be broadly different. This functionality is important, as real-time data is often noisy, distorted, and incomplete.
- ANN are *intrinsically non-linear* and can model many forms of non-linear behavior.

- ANN are *amenable to parallel execution*. With the processing requirements of large neural networks, speed is an important factor. Their structural similarity to SIMD machines makes them a viable candidates for parallel processing.

Typically, neural networks are composed of multiple layers of nodes. Figure C.1 illustrates the classification of layers into input, hidden and output layers [Hammerstrom-93]. The first and last layers are referred to as the *input* and *output* layers, respectively. Each node acts as a single processing element. Connections, are often made between successive layers in the network. Each connection has a weight associated with it that influences the data it transmits (by either strengthening or weakening its influence on the destination node).

The nodes and connections in a neural network, structurally (through the topology of the interconnections) and computationally (through node processing) implement a learning algorithm. A number of learning algorithms are in use today, each having its domains of expertise. One of the most common of these algorithms is the Back Propagation (BP) learning algorithm. Figure C.1 illustrates the initial structure and processing in a BP algorithm. Other learning algorithms include adaptive resonance theory (ART), cascade correlation (CC), learning vector quantization (LVQ), probabilistic neural network (PNN), and self-organizing (SORG) map.

Unlike computer programs, artificial neural networks are not programmed. They are taught to recognize and classify the inputs they receive. The training data set is one of the most important components of a neural network and has a direct influence on the success or failure of the network. The data set must support the inference that the trainer had in mind without narrowing its field of view, Zornetzer-90.

Training a neural network is one of the most time consuming activities in neural network development. Training time and data are often factors considered in choosing a learning algorithm.

With all these capabilities, neural networks might be assumed to be more popular then they are? However, neural networks have a number drawbacks:

- ANN may fail to find a satisfactory solution, because of a lack of data supporting a learnable function.
- The results obtained using a ANN can be very hard to account for because of the inherent non-linearity.
- ANN can be slow and expensive to train. The costs (both temporal and monetary) of data collection, analysis, and training can run very high.
- The final product may be slow, and/or inaccurate, even though technology promises better, cheaper, and faster hardware. There seems to be a non-linear relationship between performance enhancement (through the reduction of errors), and the increasing number of nodes/connections required. A common rule-of-thumb used is that the number of connections is proportional to the square of the nodes. An increase in the number of nodes in an ANN dramatically increases the number of multiply-and-accumulate operations that need to be processed, thus reducing response time.

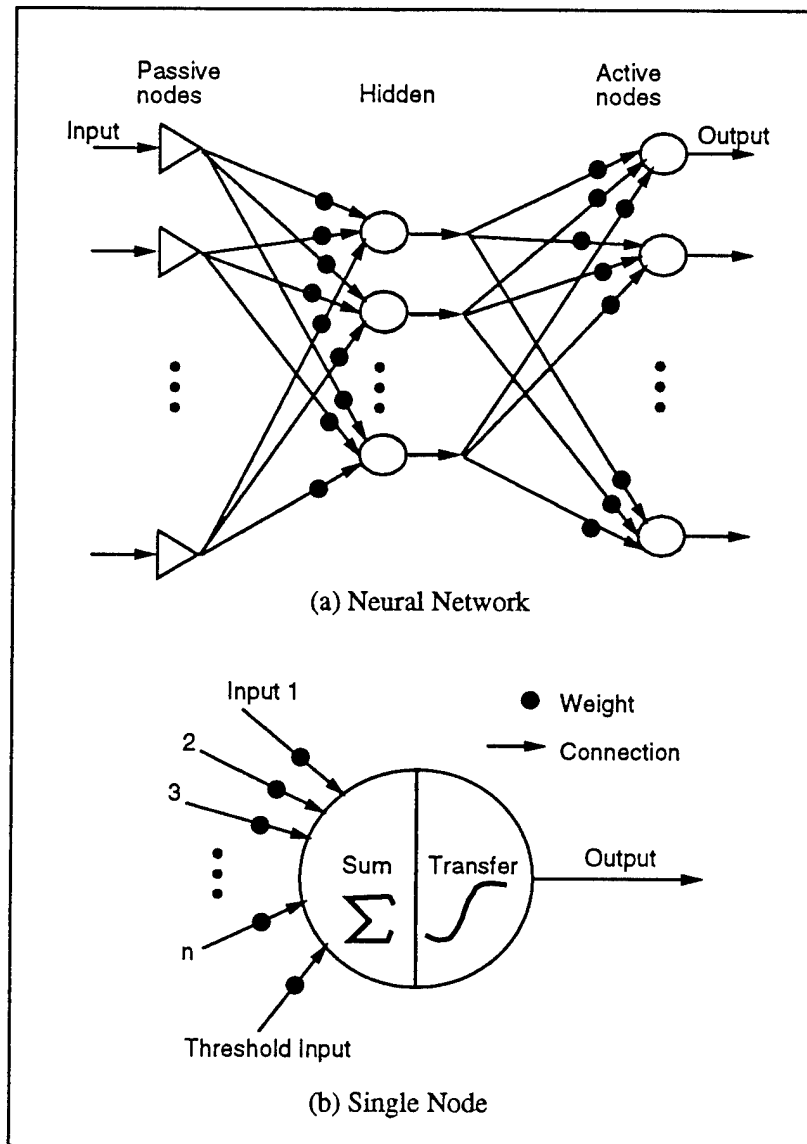


Figure C.1: A Back Propagation Neural Network

## Appendix D

# Simulation Concepts

### D.1 Real-Time Simulation

A real-time system is an information processing system which has to respond to externally generated input stimuli within a finite and specified period, [Burns-90]. Such a system must not only provide the correct logical results of computation, but must provide them in a timely manner. Hard and soft real-time constraints may be distinguished. Hard real-time constraints impose absolute deadlines; soft constraints allow deadlines to be missed occasionally depending on the workload and priorities of the system. Real-time requirements are more stringent than those found in typical interactive systems, which require as fast a response as possible but with no imposed deadlines [Allworth-87].

Real-time simulation requires that an identity correspondence be made between the time base of the simulated model and wall clock time. Moreover, all events in a model behavior stream must occur in the simulated stream at wall clock times falling within a given tolerance of their model times [Zeigler-93]. For example, the times associated with various processing functions in an HPP model (Figure D.1) would be closely emulated in the simulation. Such real-time simulation is desirable in man-in-the-loop applications, where a model of the human operator might replace one of the crew members and must respond in the same time scale as the remaining human participants so they are indistinguishable. The same real-time constraints hold when actual hardware is included in the loop : The proper timing interface must hold regardless of whether it is the hardware or the operator model that is under test.

Parallel/distributed simulation mechanisms (see Appendix D.2) are typically inapplicable to real-time simulation requirements. Mechanisms that allow component models to be processed asynchronously while maintaining correct model time relationships would not be constrained to maintain strict correspondence with wall clock time. However, the DEVS abstract simulator (see Appendix D.3) has been shown to be amenable to real-time implementation [Zeigler-93].

Special requirements on the underlying hardware and software facilitate real-time simulation.

<b>Sensory (Image Processing):</b>	Proc-Cycle = 100 [ 50–200 ] msec. Decay = 200 [ 70–1000 ] msec.
<b>Cognitive Processing:</b>	Proc-Cycle = 70 [ 25–170 ] msec. ST-Memory decay = 7 [ 5–226 ] sec.
<b>Motor Processing:</b>	Proc-Cycle = 70 [ 30–100 ] msec. Decay is the same as in ST-Memory.

Figure D.1: Requirements for Real-Time Processing

*Strict Determinism.* The same micro-operations always take the same time to be executed, regardless of the workload of the system. Such determinism makes it possible to predict the timing of macro-operations employed in the simulator.

*Sufficiently High Execution Speed.* If all micro-operations can be guaranteed to occur within tight deadlines, macro-operation timing windows can be enforced, since events can be guaranteed not to occur too late and can be delayed if necessary so as not to occur too early.

*Hardware platforms were evaluated for their support of real-time operation.*

## D.2 Approaches to Distributed Simulation

There are two basic approaches for managing processor interaction in distributed simulation algorithms: *conservative* and *optimistic*. The characteristics common to the distributed simulation are as follows, [Gimarc-89:]

- The simulation model is divided into submodels, and each submodel is assigned to a separate processor for execution.
- Each submodel has its own states and simulation clock.
- There is no shared memory, and processors communicate exclusively using messages.
- The set of processors execute concurrently until the simulation terminates.

### D.2.1 Conservative Approach

The conservative approach to parallel or distributed simulation — also called Chandy-Misra approach [Chandy-79, Chandy-81, Misra-86] — works under the following constraints.

- The couplings between the components must be static, (i.e., they must be determined before the simulation and are not allowed to change during the simulation).
- Messages sent by simulators along a channel are received in the same chronological order as they are sent.

In this approach, a simulation of a model consists of modular components which exclusively communicate through time-stamped messages obeying the causality constraint [Fujimoto-90]. The basic problem that the conservative mechanism must solve is determining when it is safe to process an event. More precisely, if a processor  $i$  contains an unprocessed event  $E_i$  with time-stamp  $T_i$  and that process can assume it will not receive another event with time-stamp smaller than  $T_i$ , the process can safely process  $E_i$ . Processes containing no safe events must block; blocking can lead to deadlock situations if appropriate precautions are not taken. There are two ways to handle deadlock.

*Deadlock Avoidance.* In this algorithm, the simulator has to prevent the possibility of a deadlock. If a cycle of blocked processes arises, then the simulation deadlocks [Peacock-79]. One scheme to avoid deadlock uses *null-messages* [Chandy-79, Misra-86] to announce the absence of regular messages.

*Deadlock Detection and Recovery.* Rather than prevent deadlock, this algorithm detects deadlock once it happens and recovers from it. Several detection and recovery methods have been proposed [Chandy-81, Misra-86, Peacock-79].

The conservative mechanism relies on good look-ahead capabilities to achieve speedup [Fujimoto-88].

Weaknesses of the conservative approach can be summarized as follow [Fujimoto-90].

- It cannot fully exploit the parallelism potential in the model.
- Its performance heavily depends on the model's look-ahead capability, (i.e., its ability to guarantee a period of safe processing in the future.)
- Its performance degradation is sensitive to model changes.
- It strongly requires static interconnections among logical processes.
- Programmers must concern themselves with details of the synchronization mechanism in order to achieve good performance.

## D.2.2 Optimistic Approach

The main optimistic mechanisms are briefly reviewed.

*Time Warp Operating System (TWOS)* [Reiher-90]. TWOS permit each node to work on its local computation at its own speed, without regard for how far ahead or behind other nodes are. It is based on the theory of virtual time [Jefferson-85].

*WOLF* [Madisetti-88]. WOLF has the advantages of limiting the error propagation to a "sphere of influence" and recovering quickly from errors. The efficiency of the algorithm depends on how well this sphere can be estimated in practice.

*Hierarchy Rollback (HR)* [Gimarc-89]. HR is a hybrid scheme containing features common to both the conservative and optimistic approaches. The three key components of HR are time management, checkpoint processing, and rollback processing. The parameterization of the checkpoint process is crucial to HR. The maximum number of checkpoint records that may be saved is a controlling factor.

The optimistic approach has focused on providing a rollback mechanism that consists of two phases [Madisetti-88].

- A Restoration Phase.
- A Cancellation Phase.

To accomplish rollback, the simulator processors have to store the following information [Praehofer-93]:

- The state information to set back the state.
- The input messages to re-do the inputs after rollback.
- An antimessage for each positive message sent to be able to annihilate the positive message in the case of rollback.

To improve performance, "lazy" versions of cancellation and reevaluation are employed [Fujimoto-90].

Weaknesses of the optimistic approach can be summarized as follows [Fujimoto-90].

- If the model contains only limited parallelism relative to the number of processors, a significant degree of rollback is inevitable.
- State-saving overhead can seriously degrade performance and space-time tradeoff if state-saving is unavoidable.
- Implementation of this approach is more complex than that of the conservative approach and may require additional hardware support to achieve significant speedup.

The DEVS formalism serves as a basis for implementing distributed simulation algorithms, [Concepcion-85, Praehofer-93, Christensen-90]. It is reviewed in the next section.

## D.3 Discrete Event System Specification

The DEVS formalism introduced by Zeigler [Zeigler-76] provides a mathematical means of defining a system composed of objects. The DEVS formalism furnishes a simple way to represent system parameters for discrete event systems. One strength of the formalism lies in its strong formal representation capable of mathematical manipulation.

DEVS-Scheme [Zeigler-90] is a knowledge-based simulation environment for modeling and design that facilitates construction of families of models in a form easily reusable by retrieval from a model base.

### D.3.1 Hierarchical, Modular Structure

The class specialization hierarchy of the DEVS-Scheme is illustrated in Figure D.2. The universal class **entities**, provides the basic constructs needed for modeling and simulation. Class **atomic-models** realize the leaf nodes (or atomic components) of the model being development under the DEVS formalism. **Coupled-models** are the major class which embodies the hierarchical model composition constructs of the DEVS formalism [Zeigler-90].

### D.3.2 The Basic Model

In building a discrete event system using the DEVS formalism, the basic models that constitute the system must first be defined. A basic model, called an atomic model (or atomic DEVS), has a specification for dynamics of the model.

The DEVS formalism defines a basic model (M) as:

$$M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta), \quad (D.1)$$

where:

$X$  = set of external input events;

$S$  = sequential state set;

$Y$  = set of external event types generated as output;

$\delta_{int} = S \rightarrow S$  : internal transition function dictating state transitions due to internal events;

$\delta_{ext} = Q \times X \rightarrow S$  : external transition function dictating state transitions due to external events; where

$Q = \{ (s, e) \mid s \in S, 0 \leq e \leq ta(s) \}$  : total state of M;

$\lambda = S \rightarrow Y$  : output function generating external events at the output;

and

$ta: S \rightarrow \text{Real-time advance function [Zeigler-84]}.$

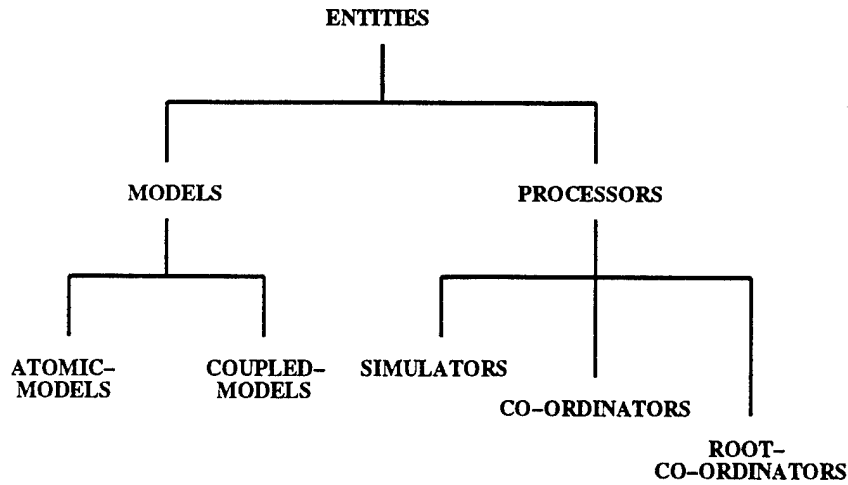


Figure D.2: Class Hierarchy of Discrete Event System Specification in Scheme

### D.3.3 The Coupled Model

The second form of the model, the *coupled model* (or coupled DEVS), provides a means of coupling (combining) several component models together to form a new model. The latter can itself be employed as a component in a larger coupled model, thus giving rise to construction of complex models in hierarchical fashion. Coupled models specify the mappings for the output to input translations and the set of influencees for each component in the coupled model.

Mathematically, a coupled model (DN) is represented:

$$DN = \langle D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, SELECT \rangle, \quad (D.2)$$

where:

$D$  is a set of *components names*;

for each  $i$  in  $D$ ,

$M_i$  is a component model,

$I_i$  is the set of influencees for  $i$ ,

and for each  $j$  in  $I_i$ ,

$Z_{i,j}$  is the  $i$ -to- $j$  output translation function

and

$SELECT = \text{subsets of } D \rightarrow D$  is the tie-breaking function [Zeigler-84].

See Figure D.3 for an example.

### D.3.4 The System Entity Structure

The System Entity Structure (SES) or entity structure is the basic means of organizing a family of possible configurations of the system being designed [Rozenblit-88]. The SES, based on a tree-like graph, is a knowledge representation scheme that combines the decomposition, taxonomy, and coupling relationships as described below.

- *Decomposition.* A way of representing a component broken down into its constituents is called decomposition. These constituents may be further broken down into subcomponents. The level of decomposition is determined by the objectives of the modeller.
- *Taxonomy.* Taxonomy provides a method for organizing information about the classification of the components and a preliminary method for categorizing systems and sub-systems into classes and subclasses through specialization.
- *Coupling.* Coupling is a representation of the knowledge about the way in which the models are connected together to form a composite model at various levels in the tree.

An SES satisfies the following axioms [Rozenblit-88, Zeigler-90].

- *Uniformity.* Any two nodes with the same names have identical attached attributes and isomorphic substructures.
- *Strict Hierarchy.* Nodes do not appear more than once down any path of the SES structure.
- *Alternating Mode.* The modes of a node and its successors are always different ("entity vs. aspect or specialization"). The root is an entity.
- *Valid Siblings.* No two nodes under the same parent (i.e., siblings) can have the same name.
- *Attached Variables.* No two variables attached to a node may have the same name.

### D.3.5 The Abstract Simulator

The architecture of DEVS Scheme simulation system is derived from the abstract simulators concepts [Concepcion-85, Zeigler-87], associated with the hierarchical, modular DEVS formalism. Since such a scheme is naturally implemented by multiprocessor architectures, models developed in DEVS Scheme are readily transportable to distributed simulation systems designed according to such principles [Christensen-90, Wang-87].

A DEVS hierarchical abstract simulator [Wang-87] consists of two types of elements: *simulators* and *coordinators*. They are assigned to handle atomic models and coupled models in a one-to-one manner, respectively. This assignment is illustrated in Figure D.4, corresponding to the model in Figure D.3. A *root-coordinator* manages the overall simulation and is linked

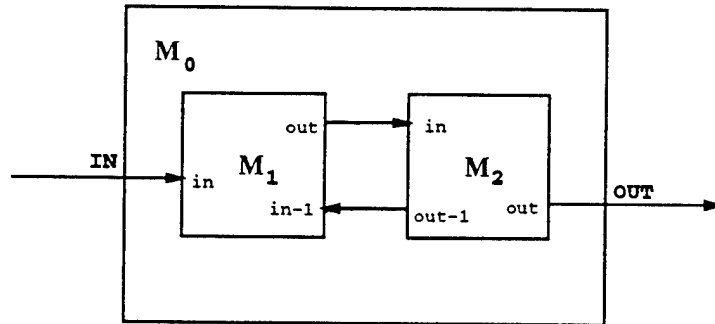


Figure D.3: Discrete Event System Specification: Concept of Modularity

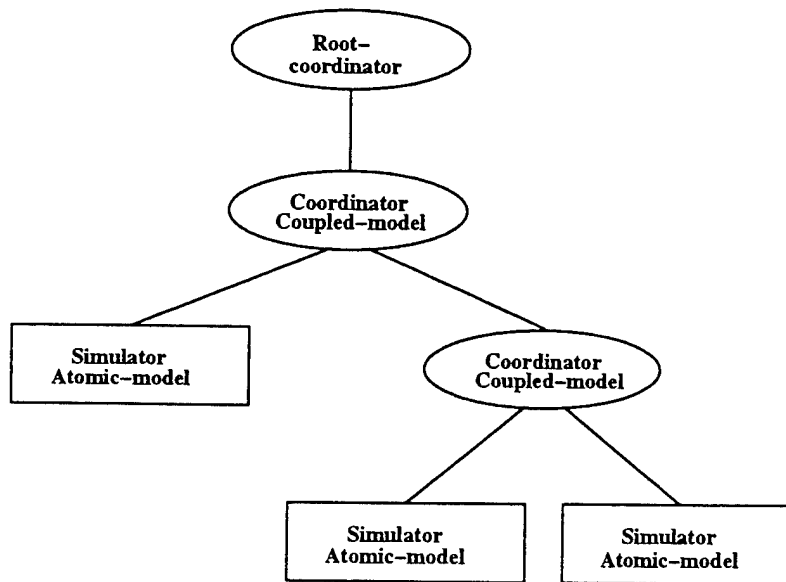


Figure D.4: Discrete Event System Specification Hierarchical Abstract Simulator Structure

```

Root-coordinator
variable:
    global clock /* global simulation clock */

Initialize:
    send message (i,t) to its child

when receive a message (d,  $t_N$ )
    if (  $t < \text{infinity}$  ) then
        clock := t
        make *_message and send message (*,t) to child
    else
        terminate

when receive a message (y,t)
    display content of this message
    passivate
end Root-coordinator

```

Figure D.5: Algorithm of Root-Coordinator

to the *coordinator* of the outermost coupled model. A simulation proceeds by passing messages among the *simulators* and *coordinators*.

The operation of an abstract simulator involves five types of messages: (\*,t), (x,t), (y,t), (i,t), and (d, $t_N$ ). The (\*,t) message asks the model to execute its output function and internal transition function. The (x,t) message carries out the external event information with a set of destinations of the influencees. The (y,t) message carries the output information sent from either a simulator or a coordinator, to its parent coordinator and the root-coordinator. The message (d, $t_N$ ) contains the "done" notice with the time of the next event  $t_N$ . These messages, along with the (i,t) message (used for initialization purposes), are exchanged among the root-coordinator, coordinators, and simulators. Algorithmic descriptions of a root-coordinator, a coordinator, and a simulator are given in Figures D.5, D.6, and D.7.

## Coordinator

variables:

$t_L$            /\* time of last event \*/  
 $t_N$            /\* time of next event \*/  
children       /\* sub-coordinators or simulators \*/  
parent         /\* parent coordinator \*/  
imminent       /\* list of imminent children \*/  
\*-child        /\* selected imminent child \*/

when receive a message (i,t)  
  for each child in children do  
    send message (i,t) to child  
  end for  
   $t_L :=$  maximum of  $t_L$  of children  
   $t_N :=$  minimum of  $t_N$  of children  
  imminent := children with minimum  $t_N$   
  \*-child := select (imminent)

when receive a message (\*,t)  
   $t_L := t$   
  send message (\*,t) to \*-child  
   $t_N :=$  minimum of  $t_N$  of children  
  imminent := children with minimum  $t_N$   
  \*-child := select (imminent)

when receive a message (x,t)  
   $t_L := t$   
  for each child in influenced children  
    send message (x,t) to child  
  end for  
   $t_N :=$  minimum of  $t_N$  of children  
  imminent := children with minimum  $t_N$   
  \*-child := select (imminent)

when receive a message (y,t)  
  if coupled to external output then  
    send message (y,t) to parent  
  for each child in influenced children  
    send message (x,t) to child  
  end for

**end Coordinator**

Figure D.6: Algorithm of Coordinator

### Simulator

variables:

```

 $t_L$       /* time of last event */
 $t_N$       /* time of next event */
 $e$         /* elapsed time in this state */
 $s$         /* state of the model component */
 $x$         /* external input */
 $y$         /* output from model component */
parent    /* parent coordinator */
```

when receive a message ( $x, t$ )

```

  if  $t_L \leq t \leq t_N$  then
     $e := t - t_L$ 
     $s := \delta_{ext}(s, e, x)$ 
     $t_L := t$ 
     $t_N := t_L + ta(s)$ 
    send message ( $d, t_N$ ) to parent
  else
    error
```

when receive a message ( $*, t$ )

```

  if  $t = t_N$  then
     $y := \lambda(s)$ 
    send ( $y, t$ ) to parent
     $s := \delta_{int}(s)$ 
     $t_L := t$ 
     $t_N := t_L + ta(s)$ 
    send message ( $d, t_N$ ) to parent
  else
    error
```

when receive a message ( $i, t$ )

```

   $t_L := t$ 
   $t_N := t_L + ta(s)$ 
end Simulator
```

Figure D.7: Algorithm of Simulator

## Appendix E

# Examples of Human Performance Process Hierarchical Decomposition

This appendix provides an example of a framework for a hierarchically expandable HPP model. A Distributed Cognitive Simulation Architecture model (illustrated in Figure E.1), initially modeled on a single workstation, is expanded to a distributed network. The distributed model could use a network of three workstations connected in a ring topology, where each workstation is used to model a single sub-system.

Figures E.1— E.7 provide further decompositions and alternatives for hierarchical expandability of HPP models.

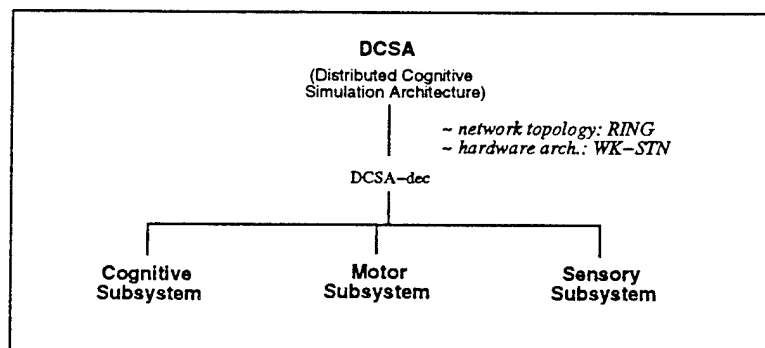


Figure E.1: Distributed Cognitive Simulation Architecture (Root Node)

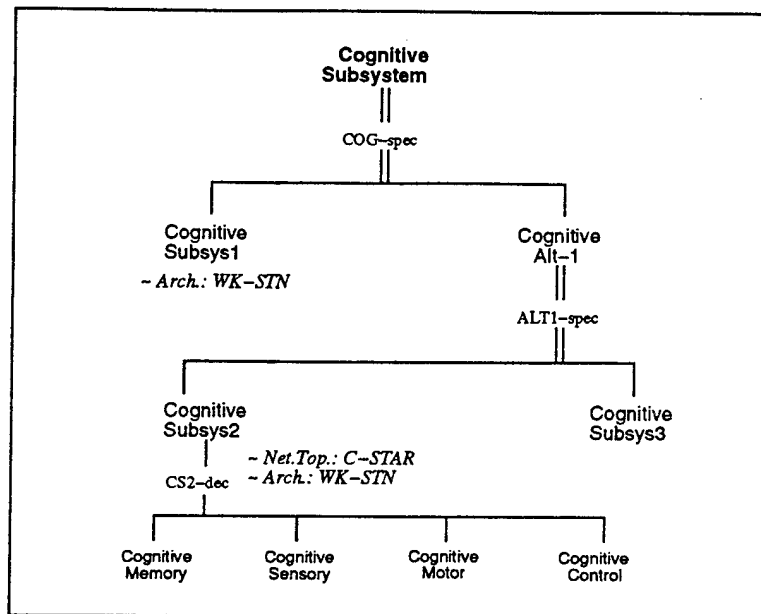


Figure E.2: Cognitive Subsystem Decomposition

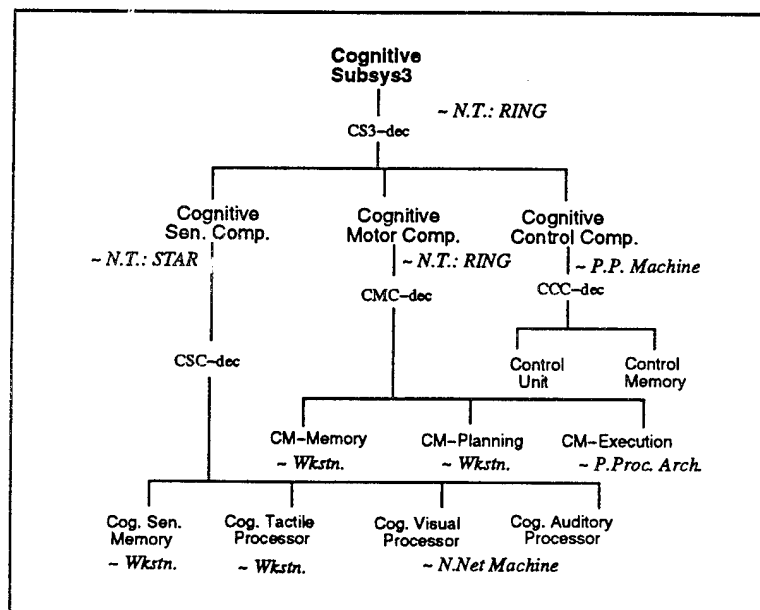


Figure E.3: Cognitive Subsystem 3 Decomposition

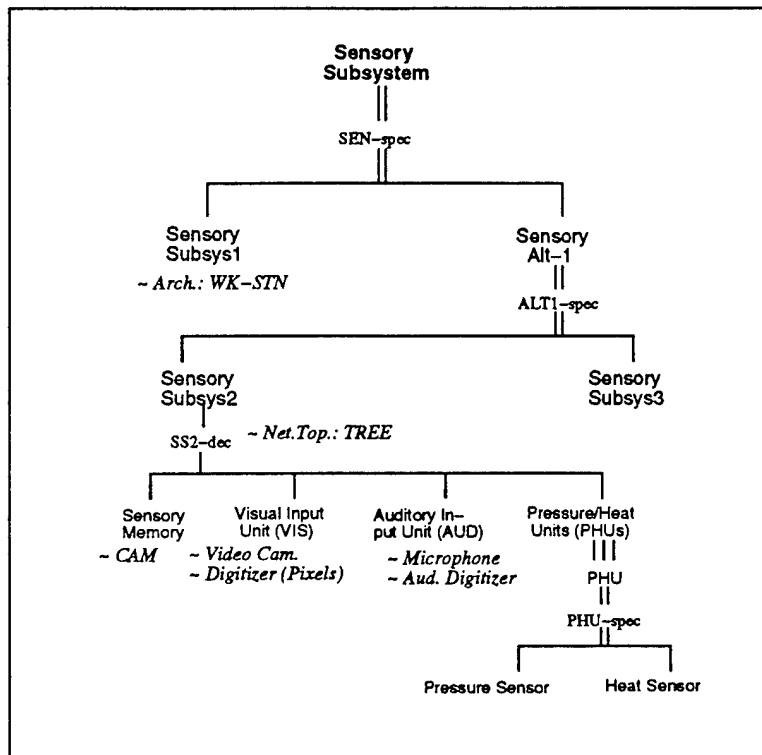


Figure E.4: Sensory Subsystem Decomposition

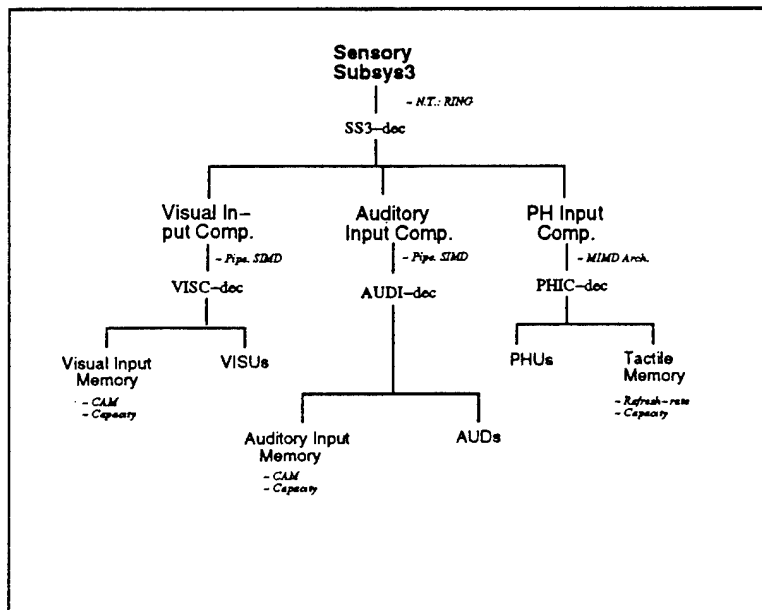


Figure E.5: Sensory Subsystem 3 Decomposition

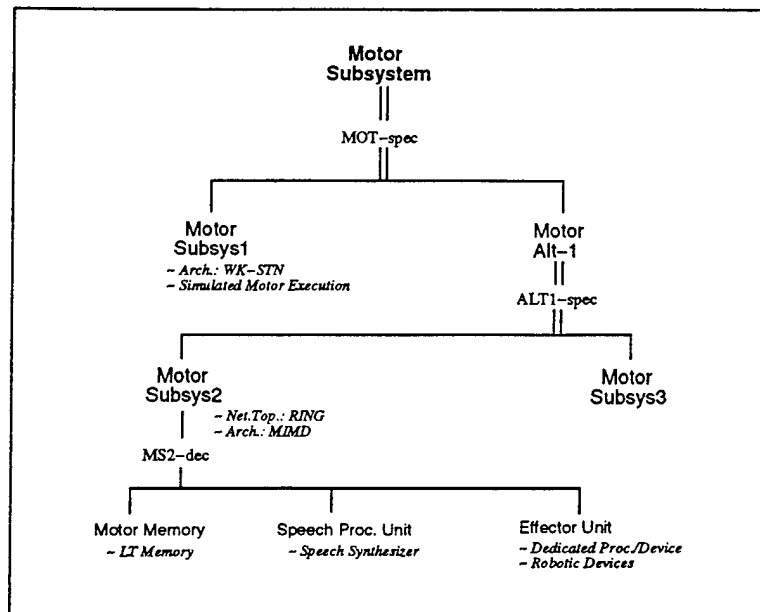


Figure E.6: Motor Subsystem Decomposition

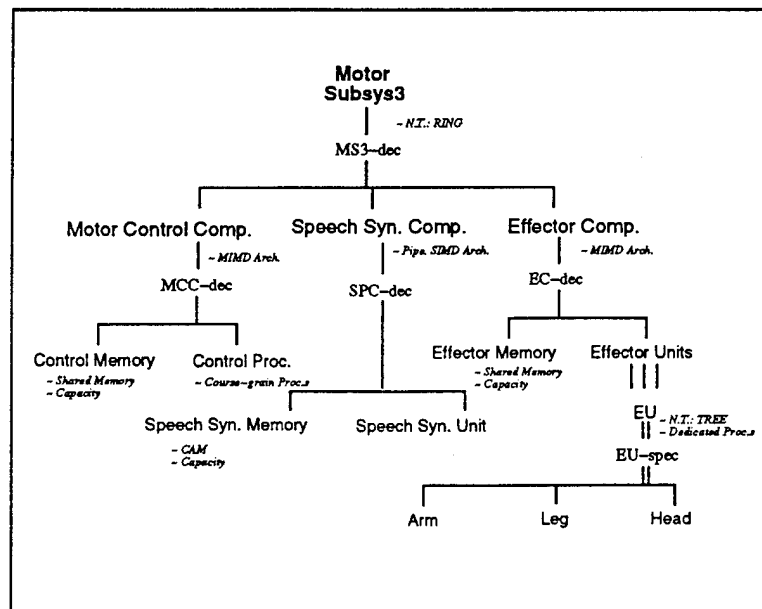


Figure E.7: Motor Subsystem 3 Decomposition

## Appendix F

# Scalable Coherent Interfaces

Figure F.1 compares the new SCI interconnect standard with the current market leader, VME Bus and the proposed enhancement Futurebus+. The data taken from Berube & Anderson shows that SCI (and its real-time version SCI/RT) will be capable of significantly higher performance, flexibility, and fault tolerance than its competitors. A commercial implementation demonstrating its potential feasibility, for hierarchically expandable simulation under both real-time and non real-time requirements has recently been announced [Berube-93].

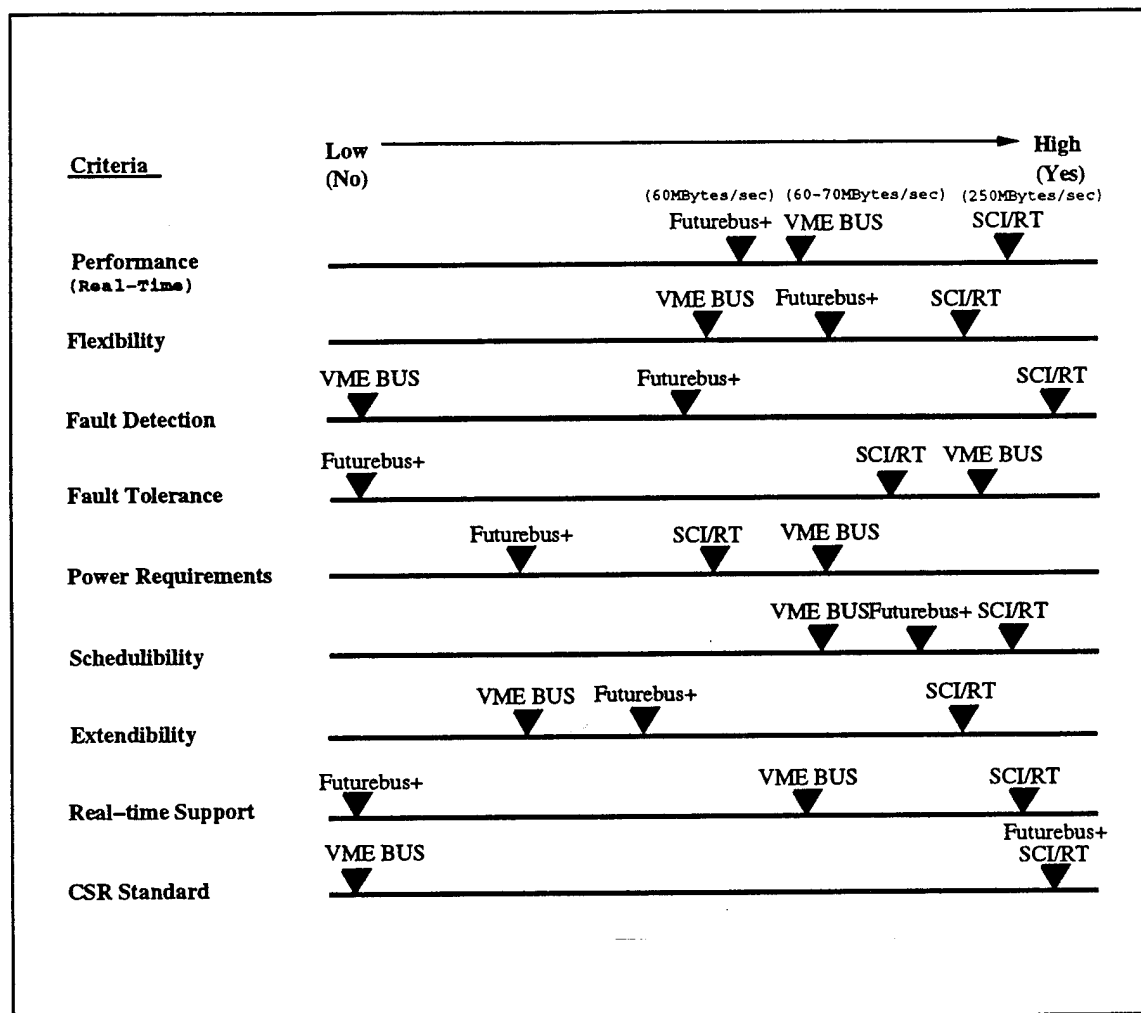


Figure F.1: Evaluation of Network Interconnection Interfaces

## Appendix G

# Synopses of Selected High Performance Parallel Machines

- CM-5: 1. **Style:** synchronized MIMD.
2. **Processors:** make = SPARC, speed = 32 MHZ, memory = 32 MBytes, speed = 128 MFLOPS, word = 64 bits for vector processing unit, configuration of 32 to 16K processing nodes, control processor consists of a RISC microprocessor (CPU), memory, I/O with local disks and Ethernet connections, and a network interface.
  3. **Topology:** Fat tree for Data Network, Universal Architecture (Data, Control, and Diagnostic Network).
  4. **Memory:** 8, 16, or 32 MBytes of DRAM for processing nodes.
  5. **Programming Model:** Extended Data Parallel model.
  6. **Cache Coherence:** 64 KBytes of cache provided for each processing node.
  7. **Synchronization:** control processors supervises processing nodes, block broadcasting, branching.
  8. **Performance:** max 59 GFlops for 1K processing nodes system.
  9. **Unique:** fault-tolerance (error-detection circuitry in processor), TFlop performance, hardware support for both SIMD and MIMD, very high communication, memory, and I/O bandwidth.
  10. **Languages:** FORTRAN90, \*LISP, C\*, Paris.
  11. **I/O:** Data Vault(up to 60 GBytes) with sustained rate of 25 MBytes/s, multiple CM-HIPPI(I/O controller with 8 CMIO ports), CM-IOP with 16 ports.
  12. **OS:** CMOST; enhanced version of Unix, support for timesharing and batch processing(NQS), partition management.

**CM-200:** 1. Style: SIMD.

2. **Processors:** optional floating-point accelerator (Weitek chip plus interface). each with  $2^{16}$  bits of addressable memory, word size = 32 bits, make = custom-bit processor, speed = CM-2 7MHZ, CM-200 10MHZ, up to 64K processors.
3. **Topology:** Fat tree exploiting the locality of reference.
4. **Memory:** 4 MByte per Weitek processor.
5. **Programming Model:** Data Parallel.
6. **Cache Coherence:** no requirement.
7. **Synchronization:** special hardware support, broadcasting, global combining, scalar memory bus.
8. **Performance:** max 9 GFlops.
9. **Unique:** Fault-tolerance (error-detection circuitry in processor), virtual processor mode.
10. **Languages:** FORTRAN90, \*LISP, C\*, Paris.
11. **I/O:** max 320 MBytes/S transfer rate with 8 I/O controllers, Data Vault.
12. **OS:** compatible with Unix or LISP.

**MP-1:** 1. Style: SIMD.

2. **Processors:** up to 16K 4-bit processors, 40 programmable 32-bit registers, word size = 64 bits, make = custom-chip, speed = a few dozen KFlops.
3. **Topology:** X-Net (multistage crossbar interconnection network).
4. **Memory:** 16KBytes per processor, implemented using page mode 1MBit DRAMs. max 1 GBytes.
5. **Cache coherence:** local memory provided.
6. **Synchronization:** global bus connecting the array control unit to all processors.
7. **Performance:** MP-2 = max 1.6 GFlops MP-1 = max 26 GIPS(32-bit operations) or 550 MFlops (double-precision).
8. **Languages:** FORTRAN, C, and MasPar Parallel Application Language (MPL).
9. **Unique:** all communication paths are equal lengths and arrive at destinations simultaneously.
10. **OS:** ULTRIX runs on the host and provides a standard user interface.
11. **I/O:** bandwidth = 12 GBits / sec for inter-PE communication bandwidth = 23 GBits/sec with X-Net.

**nCUBE-2S:** 1. Style: MIMD.

2. **Processors:** 8-8192, word size = 64 bits, speed = 25 MHZ, 4.1 MFlops (32bit) 3.0 MFlops (64bit), make = proprietary, memory = 4 - 64 MBytes.
3. **Topology:** Hypercube.
4. **Main Memory:** 32 MBytes - 262 GBytes 0 - 7141 GBytes for disk storage.

5. **Cache Coherence:** distributed memory architecture, thus, no cache coherence problems.
6. **Bandwidth:** processor-to-Memory = 800 MBytes/s - 819 GBytes/s Interprocessor Communication = 572 MBytes/s - 577 GBytes/s.
7. **Synchronization:** single system clock.
8. **Performance:** max 34 GFlops.
9. **Unique:** customized VLSI chip design approach.
10. **Languages:** sequential, parallel, and SPMD programming model, FORTRAN, C and C++ DBMS (Oracle).
11. **I/O:** UNIX I/O interface extended for parallel, scalable I/O and transparent disk stripping, equipped with nCHANNEL, HIPPI, nVision Real-Time Graphics boards.
12. **OS:** nCX provides UNIX System V.4 system-call interface and POSIX signals.

**KSR1:** 1. **Style:** MIMD.

2. **Processors:** 8-1088 with private 32 MBytes caches, word size = 64 bits, make = proprietary, speed = 40 MFlops.
3. **Topology:** hierarchical fat-tree (bandwidth increasing with level to 4 GBits/s) linking caches, scalable.
4. **Memory:** shared, logical view=single  $2^{64}$  byte address space.
5. **Cache Coherence:** unit of information is a sub-page = 128 Bytes = Cache line, copy or invalidate cache lines except for very large objects, 2/3 of hardware manages the memory and 1/3 does the processor.
6. **Synchronization:** data locks, barriers, critical regions, condition variables.
7. **Performance:** max 43 GFlops.
8. **Unique:** a complete shared memory programming model enables the vendor to provide a full native UNIX system and applications programmers to port and parallelize programs changing only about 1% of the source code. Exploits locality of reference; will run message passing programs faster since hierarchy is built in.
9. **Languages:** DBMS (ORACLE), COBOL, 4GLS, CASE, GUI tools, FORTRAN, C, C++.
10. **I/O:** Asynchronous, caching and scalable disk I/O.
11. **OS:** compatible with System V.3 and V.4, 4.3BSD and 4.4BSD POSIX 1003.1, XPG4.

**GigaCube:** 1. Style: MIMD.

2. **Processors:** T9000 integrated with 32 bit processor, a 64 bit floating point unit, 16 KBytes Cache, a communication processor and four links, word size = 64 bits, make = Inmos, speed = 200 MIPS, 25 MFlops and transmission rates of 8\*100 MBits/s at 50 MHZ.
3. **Topology:** physical = 3 dimensional array of up to 16\*32\*32 virtual = e.g. two or four dimensional arrays, trees or hypercubes.
4. **Memory:** 17 processors in one cluster, each processing node with 32 MBytes.
5. **Programming model:** coarse-grained parallelism well supported by sequential languages.
6. **Synchronization:** global event synchronization mechanism.
7. **Performance:** max 400 GFlops(190 GFlops sustained, double-precision).
8. **Unique:** link and memory monitoring, EDC and memory scrubbing function, three available communications = synchronous link-bound, asynchronous link-bound, asynchronous mailbox communication Component Distribution Language (CDL) taken from Helios OS, used to describe distributed programs.
9. **Languages:** FORTRAN, C, ANSI-C, Pascal, Modula-2, and Occam
10. **I/O:** each cube has 640 MBytes/sec using special C104s.
11. **OS:** Parix based on Unix front-end.

# Appendix H

## Glossary of Acronyms

### Chapter 1 :

ACT: Adaptive Control of Thought  
AI: Artificial Intelligence  
BBN: Bolt, Baranek & Newman  
CONOPS: Concept of Operations  
DP: Data Processing  
HPM: Human Performance Modeling  
HPP: Human Performance Process  
USAF: United States Air Force

### Chapter 2 :

ADM: Adaptive Decision Maker  
DAI: Distributed Artificial Intelligence  
DOS: Distributed Operating Systems  
EP: Engineering Psychology  
FM: Fallible Machine  
HAS: High Autonomy Systems  
IR: Intelligent Robotics  
ND: Neural Darwinism  
SOM: Society of Minds

### Chapter 3 :

PRP: Psychological Refractory Period  
RT: Reaction Time

### Chapter 4 :

CLOS: Common Lisp Object System  
DEVSS: Discrete Event Simulation System  
GPSS: General Purpose Simulation System  
KEE: Knowledge Engineering Environment  
OMAR: Operator Model ARchitecture

OO: Object-oriented  
SCORE: Simulation CORE language  
SPROCKET: Simulation & PROcedure Knowledge Editing Tools

**Chapter 6 :**

ANN: Artificial Neural Networks  
CM: Connection Machine  
DARPA: Defense Advanced Research Project Agency  
MIMD: Multiple Instruction Multiple Data  
MISD: Multiple Instruction Single Data  
NAS: Numerical Aerodynamics Simulation  
SIMD: Single Instruction Multiple Data  
SISD: Single Instruction Single Data

**Chapter 8 :**

CPU: Central Processing Unit  
HC: Heterogeneous Computing  
Hippi: High Performance Parallel Interface  
I/O: Input-Output  
LAN: Local Area Network  
RTOS: Real-Time Operating System  
SCI: Scalable Coherent Interface

**Chapter 9 :**

HCE: Heterogeneous Computing Environment  
MPP: Massively Parallel Platforms  
OP: Operational Peak Performance  
PC: Personal Computer  
PE: Processing Elements  
RAM: Random Access Memory  
TP: Theoretical Peak Performance

**Chapter 10 :**

SES: System Entity Structure  
SCI: Scalable Coherent Interface

**Appendix A :**

FWM: Focal Working Memory  
KB: Knowledge Base  
POC: Performance Operating Characteristics  
PRF: Performance Resource Characteristics  
PWF: Peripheral Working Memory  
ROC: Receiver Operating Characteristics

**Appendix B :**

CLIM: Common List Interface Manager  
GPP: Goal-Plan-Procedure  
KEE: Knowledge Engineering Environment  
VSFL: Very Simple Frame Language

**Appendix C :**

ART: Adaptive Resonance Theory  
BP: Back Propagation  
CC: Cascade Correlation  
LVQ: Learning Vector Quantization  
PNN: Probabilistic Neural Network  
SORG: Self Organizing Map

**Appendix D :**

HR: Hierarchical Rollback  
TWOS: Time Warp Operating System

**Appendix E :**

DCSA: Distributed Cognitive Simulation Architecture

## References

- [Abrett-70 ] G. Abrett, S. Deutsch, and S. Downes-Martin. Two AI languages for simulation. *Trans. of the Soc. for Computer Simulation*, 7(3), 1970.
- [Adaptive-93 ] Adaptive Solutions. The C-NAPS Server. Technical report, Adaptive Solutions Inc., 1993.
- [Ahuja-86 ] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *IEEE Computer*, pages 26-34, August 1986.
- [Albus-90 ] J. S. Albus. A theory of intelligent systems. In *5th IEEE Symposium on Intelligent Control*, pages 866-875, 1990.
- [Allworth-87 ] S. T. Allworth and R. N. Zobel. *Introduction to Real-Time Software Design*. Macmillan Education Foundation, New York, 1987.
- [Anderson-90 ] J. Anderson. *The Adaptive Character of Thought*. Lawrence Earlbaum Assoc., Hillsdale, N.J., 1990.
- [Antsaklis-89 ] P. J. Antsaklis, K. M. Passino, and S. J. Wang. Towards intelligent autonomous control systems: Architecture and fundamental issues. *J. Intell. Robot. Syst.*, 1(4): 315-342, 1970.
- [Arbib-89 ] M. Arbib. *The Metaphorical Brain 2: Neural Networks and Beyond*. John Wiley and Sons, New York, 1989.
- [Arnould-91 ] E. Arnould. The design of Nectar: A network backplane for heterogeneous multi-computers. *Proc. Intl. Conf. on Arch. Supp. for OS and PL*, 1991.
- [Bailey-93 ] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon. Nas parallel benchmark results. *IEEE Parallel and Distributed Technology*, pages 43-51, February 1993.
- [Balci-86 ] O. Balci. Requirements for model development environments. *Computers and Operations Research*, 13(1): 53-67, 1986.
- [Balmer-86 ] D. Balmer. CASM— the right environment for simulation. *J. Operations Research Soc.*, 37:443-452, 1986.
- [Baron-90 ] S. Baron, D. Kruser, and B. Huey. *Quantitative Modelling of Human Performance in Complex, Dynamic Systems*. National Academy Press, Washington, D.C., 1990.
- [Berube-93 ] L. Berube and D. Anderson. A discussion of the functionality and performance of the IEEE p1596.6 scalable coherent interface for real-time applications (SCI/RT) proposed draft standard. Technical report, Edgewater Computer Systems, Inc., May 1993.
- [Burns-90 ] A. Burns and A. Wellings. *Real-Time Systems and their Programming Languages*. Addison-Wesley Co., NY, 1990.
- [Caudill-87 ] M. Caudill. Neural networks primer: Part 1. *AI EXPERT*, pages 46-52, December 1987.

- [Chandy-79 ] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed systems. *IEEE Trnas. on Software Eng.*, 5(5): 440-452, 1979.
- [Chandy-81 ] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11), November 1981.
- [Christensen-90 ] E. R. Christensen and B. P. Zeigler. Distributed discrete event simulation: Combining DEVS and Time Warp. In *Proc. the SCS Multiconference on Distributed Simulation*, pages 148-152, 1990.
- [Concepcion-85 ] A. I. Concepcion. Mapping distributed simulators onto the hierarchical multibus multiprocessor architecture. In *Proc. of the Distributed Simulation Conference*, San Diego, CA., 1985.
- [Damasio-89 ] A. R. Damasio. Time-locked multiregional retroactivation: A systems-level proposal for the neural substrates of recall and recognition. *Cognition*, 33: 25-62, 1989.
- [Dongarra-93 ] J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, CS Department, University of Tennessee, March 1993.
- [Dongarra-93a ] J. Dongarra, H. W. Meuer, and E. Strohmaier. Top500 supercomputers. Technical report, CS Department, University of Tennessee and University of Mannheim, July 1993.
- [Edelman-87 ] G. Edelman. *Neural Darwinism*. Basic Books Inc., New York, 1987.
- [Elkind-89 ] J. Elkind et. al. *Human Performance Models for Computer-Aided Engineering*. National Academy Press, Washington, D.C., 1989.
- [Fishwick-90 ] P. Fishwick. Towards an integrated approach to simulation model engineering. *J. General Systems Resrach*, 17(1): 1-20, 1990.
- [Freund-93 ] R. Freund and H. J. Siegel. Heterogeneous processing. *IEEE Computer*, pages 13-17, June 1993.
- [Fujimoto-88 ] R. M. Fujimoto. Lookahead in parallel discrete event simulation. In *Proc. of the International Conference on Parallel Processing*, pages 34-41, Silver Springs, MD, 1988.
- [Fujimoto-90 ] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33: 30-53, 1990.
- [Furtney-93 ] M. Furtney and G. Taylor. Of workstations and supercomputers. *IEEE Spectrum*, pages 50-58, May 1993.
- [Gimarc-89 ] R. L. Gimarc. Distributed simulation using hierarchical rollback. In *Proc. of the Winter Simulation Conference*, pages 621-629, 1989.
- [Goldberg-83 ] A. Goldberg and R. David. *Smalltalk-80: The Language and its Application*. Addison Wesley, Reading, MA., 1983.
- [Hammerstrom-93 ] D. Hammerstrom. Working with neural networks. *IEEE Spectrum*, pages 46-53, July 1993.
- [Hayes-Roth-83 ] F. Hayes-Roth and et al., editors. *Building Expert Systems*. Addison Wesley Pub., Reading, MA., 1983.
- [Henriksen-83 ] J. Henriksen. The integrated simulation environment. *Operations Research*, 31: 1053-1073, 1983.
- [Hudlicka-92 ] E. Hudlicka and et. al. Computational cognitive models- informal technical summary. Technical Report 7752, Bolt, Barneck & Newman Inc., 1992.
- [Hwang-84 ] K. Hwang and F. Briggs. *Computer Architecture and Parallel Processing*. McGraw Hill, New York, 1984.

- [Jefferson-85 ] D. R. Jefferson. Virtual time. *ACM Trans.on Programming Languages and Systems*, 7(3): 404-425, 1985.
- [Jhonson-93 ] R. E. Jhonson. *Extending The Scalable Coherent Interface for Large-scale Shared-Memory Multiprocessors*. PhD thesis, University of Wisconsin-Madison, 1993.
- [Kaashoek-89 ] M. F. Kaashoek, H. E. Bal, and A. S. Tanenbaum. Experience with the distributed data structure paradigm in linda. In *Distributed and Multiprocessor Systems Workshop Proceedings*, pages 175-191. USENIX, October 1989.
- [Khokhar-93 ] A. A. Khokhar and et al. Heterogeneous computing: Challenges and opportunities. *IEEE Computer*, pages 18-27, June 1993.
- [Klahr-86 ] P. Klahr. Expressibility in ross, an object-oriented simulation system. In *Artificial Intelligence in Simulation*. SCS Publishers, San Diego, CA., 1986.
- [Koestler-67 ] A. Koestler. *The Ghost in the Machine*. Cambridge U. P., Cambridge, U.K., 1967.
- [Koestler-78 ] A. Koestler. *Janus: The Summing Up*. Hutchinson Pub., London, U.K., 1978.
- [Kosslyn-80 ] S. Kosslyn. *Image and Mind*. Harvard U. P., Cambridge, Ma., 1980.
- [Leler-90 ] W. Leler. Linda meets UNIX. *IEEE Computer*, 23(4): 43-54, February 1990.
- [Lesser-90 ] V. Lesser. An overview of distributed AI. *J. Japanese Soc. of Artif. Intell.*, 5: 392-400, 1990. Special issue on Distributed AI.
- [Madisetti-88 ] V. Madisetti, J. Walrand, and D. Messerschmitt. Wolf: A rollback algorithm for optimistic distributed simulation systems. In *Proc. of the Winter Simulation Conference*, pages 296-305, 1988.
- [Minsky-86 ] M. Minsky. *The Society of the Mind*. Simon and Schuster, New York, 1986.
- [Misra-86 ] J. Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1): 31-65, 1986.
- [Morgan-93 ] P.D. Morgan. Simulation of an adaptive behavior mechanism in an expert decision maker. *IEEE Trans. Sys. Man & Cyber.*, 23(1): 65-73, 1993.
- [Murray-87 ] K. Murray and S. Shepherd. Automatic synthesis using automatic programming and expert systems techniques toward simulation modeling. In *Proceedings of the Winter Simulation Conf.*, 1987.
- [Newell-90 ] A. Newell. *Unified Theories of Cognition*. Harvard U. P., Cambridge, Ma., 1990.
- [Oren-79 ] T. Oren and B. P. Zeigler. Concepts for advanced simulation methodologies. *Simulation*, 32(3): 69-82, 1979.
- [Palmucci-92 ] J. Palmucci. A simulation core language. BBN Inc.— Internal Report, 1992.
- [Peacock-79 ] J. K. Peacock, J. W. Wong, and E. G. Manning. Distributed simulation using a network of processors. *Computing Networks*, 3(1): 471-474, 1979.
- [Praehofer-93 ] H. Praehofer. Distributed discrete event simulation. Technical Report TR-93, University of Linz, 1993.
- [Rajkumar-89 ] R. Rajkumar, L. Sha, and J. P. Lehoczky. Futurebus+ in real-time systems. Technical report, Carnegie Mellon University, June 1989.
- [Ramanathan-93 ] G. Ramanathan and J. Oren. Survey of commercial parallel machines. Technical Report TR-CS-93-70-4, CS Department, Oregon State University, March 1993.
- [Rasmussen-86 ] J. Rasmussen. *Information Processing and Human Computer Interaction*. North-Holland Pub., New York, 1986.
- [Reason-90 ] J. Reason. *Human Error*. Cambridge U. P., Cambridge, U.K., 1986.
- [Reddy-86 ] Y. Reddy and et al. The knowledge-based simulation system. *IEEE Software Eng.*, pages 26-37, March 1986.

- [Reiher-90 ] P. L. Reiher. Parallel simulation using the time warp operating system. In *Proc. of the Winter Simulation Conference*, pages 38–45, 1990.
- [Rozenblit-88 ] J. W. Rozenblit and B. P. Zeigler. Design and modelling concepts. *International Encyclopedia of Robotics Applications and Automation*, pages 308–322, 1988.
- [Rozenblit-90 ] J. W. Rozenblit and et al. Knowledge-based Design and Simulation Environment (KDSE): Foundational concepts and implementation. *J. Operations Research Soc.*, 41(6): 475–489, 1990.
- [Ruiz-89 ] S. Ruiz-Mier and J. Talavage. A hybrid paradigm for modeling of complex systems. In *Artificial Intelligence, Simulation and Modelling*. J. Wiley Publishers, New York, 1989.
- [Saridis-83 ] G. N. Saridis. Intelligent robotic controls. *IEEE Trans. Autom. Control*, AC-28(5), 1983.
- [Sha-93 ] L. Sha, R. Rajkumar, and S. S. Sathaye. Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. To be published in IEEE Proceedings Journal, February 1993.
- [Shu-90 ] D. Shu, J. Nash, and C. Weems. A multiple-level heterogenous architecture for image understanding. *Proc. Intl. Conf. Pat. Rec.*, 1990.
- [Tanenbaum-92 ] A. S. Tanenbaum. The amoeba distributed operating system. Vrije Universiteit, email: ast@cs.vu.nl, 1992.
- [Tanir-93 ] O. Tanir and S. Sevinc. Defining the requirements for a standard simulation environment. *IEEE Computer*, December 1993.
- [VanLehn-91 ] K. VanLehn, editor. *Architectures for Intelligence*. Lawrence Earlbaum Assoc., Hillsdale, N.J., 1991.
- [Wang-87 ] Y. H. Wang. The implementation of the hierarchical abstract simulator on the iPSC computer. Master's thesis, University of Arizona, 1987.
- [Wickens-84 ] C. Wickens. *Engineering Psychology and Human Performance*. Charles E. Merrill, Columbus, Oh., 1984.
- [Young-92 ] M. Young. A cognitive architecture for human performance process model research. Technical Report AL-TP-1992-0054, Air Force Materiel Command, Wright-Patterson AFB, Ohio, 1992.
- [Young-93 ] M. Young. Successively approximating human performance. Technical Report AL-TP-1993-0026, Air Force Materiel Command, Wright-Patterson AFB, Ohio, 1993.
- [Zeigler-76 ] B. P. Zeigler. *Theory of Modelling and Simulation*. Wiley, New York, 1976.
- [Zeigler-84 ] B. P. Zeigler. *Multifaceted Modelling and Discrete Event Simulation*. Academic Press, Orlando, Fl., 1984.
- [Zeigler-87 ] B. P. Zeigler. Hierarchical, modular discrete event modeling in an object oriented environment. *Simulation J.*, pages 219–230, 1987.
- [Zeigler-90 ] B. P. Zeigler. *Object-Oriented Simulation with Hierarchical, Modular Models*. Academic Press, San Diego, Ca., 1990.
- [Zeigler-91 ] B. P. Zeigler and C. Luh. Model base management for multifaceted systems. *ACM Trans. on Modelling and Comp. Sim.*, 1(3): 195–218, 1991.
- [Zeigler-93 ] B. P. Zeigler and J. Kim. Extending the devs-scheme knowledge-based simulation environment for real-time event-based control. *IEEE Trans. on Robotics and Automation*, 1993.
- [Zeigler-95 ] B. P. Zeigler, M. Young, and S. Vahie. Successive approximation in multifaceted modelling methodology: Human performance application. *Int. Journal of Simulation*, 1995.

To appear.

[Zornetzer-90 ] S. Zornetzer and et al., editors. *Introduction to Neural and Electronic Networks*.  
Academic Press, 1990.